

```

; This is an monitor for the S100computers.com 8086 board (and later 8088,80286 boards)

; It started from the simple monitor in Byte on Nov 1980 but is enlarged and

; modified to work with the S100Computers 8086 Board, IDE Board, ZFDC board, MSDOS Support (PIC & RTC) Board and other

; hardware as well. More recently it has been extensively enlarged to contain the interrupt based functions required

; to run a Microsoft's MS-DOS (V4.01) or FreeDOS emmulating an IBM-PC BIOS ROM.

;

;   John Monahan, San Ramon, CA   (monahan@vitasoft.org)

;   History

;   V1.0           ;Original version sometime in 1982

;   V2.1   3/12/1983

;   V2.2   11/12/09   ;Modified for simple I/O. Ports info command added

;   V2.3   11/18/09   ;Reset FAR jump to start of monitor, added Register display

;   V2.31  11/19/09   ;Allow intersegment FAR jump with G command

;   V2.4   2/18/10           ;Write version to reside at F:F000H (to be loaded with a CPM loader from disk)

;   V2.5   8/26/10           ;Add S100Computers Serial IO Board & RTC Board. Input IOBYTE (EFh) for JMP to 0:500H

;   V2.51  8/26/10           ;Stack & Flag below EPROM in high memory

;   V2.52  8/27/10           ;Stack etc in low memory. AP/PM for clock (with DAS opcode)

;   V2.53  8/27/10           ;Check if after a reset a direct jump to CPM86 in RAM is required (rather than this monitor).

;   V2.6   5/7/11           ;Added IDE Board diagnostic package

;   V2.7   5/9/11           ;Aruba trip, complete overhaul while traveling.

;   V2.8   5/14/11           ;Finished IDE drive additions

;   V2.9   5/17/11           ;Switched over to using SI and DI registers for memory move etc functions

;   V3.0   5/31/11           ;Corrected Sector display routines

;   V3.1   6/1/11           ;Corrected memory Map display and move memory routines

```

```

;   V3.2   6/7/11      ;Corrected CF Drive A=B Verify routine
;   V3.3   6/8/11      ;Corrected CF Drive A->B copy routine
;   V3.4   6/8/11      ;BP used for all IDE routines thus freeing dependance on a fixed RAM location
;   V3.5   6/10/11     ;Corrected Disk format routine. Corrected Drive ID routine
;   V3.6   6/19/11     ;Corrected Monitor signon message at start
;   V4.0   7/20/11     ;Splice in IBM-PC/MS-DOS Interrupt routines. Enlarged Monitor now starts at FC000H
;   V4.1   7/31/11     ;Correct CICO routine so it is not case sensitive
;   V4.2   8/3/11      ;Vector Int's 0 & 1 working OK.
;   V4.3   8/4/2011    ;MS-DOS 2.01 loading from floppy fine on 5" SS Disks (only)!
;   V4.4   8/20/2011   ;Added special MS-DOS FFDC commands to read DDDS DOS Disks. Now works with IBMs PC-DOS up to V3.1
;   V4.51  8/23/2011   ;Corrected length check with GET5DIGIST etc.
;   V4.52  8/23/2011   ;Added "PATCH" to quickly test RAM/Debug versions of this code.
;   V5.0   8/26/2011   ;MS-DOS hard disk caspability
;   V5.1   9/1/2011    ;Corrected bug in IDE (WR_LBA) routine. Was not sending High Cylinder! (not used with CPM!)
;   V5.2   9/6/2011    ;Move to 27C256 EEPROMS (Will no longe fit in 27C64's). Address starts at F000:8000H
;   V5.3   9/7/2011    ;Last version written for Digital Research ASM86 assembler. (Too big, get symbol overflow)
;   V6.0   9/8/2011    ;Major rewrite to work with NASM Assembler. (Sorry I did not do this earlier. Much better Assembler)
;   V6.1   9/11/2011   ;Corrected IDE disk compare routine.
;   V6.2   9/12/2011   ;Added IDE memu options to test LBA & CHS display on IDE Board HEX display LED's
;   V6.3   9/13/2011   ;Added IBM-BIOS menu option the Read/Write a block of contiguous sectors the the IDE Drive
;   V6.4   9/16/2011   ;Added cursor addressing to video output functions. FDISK now displays correctly.
;   V6.5   9/17/2011   ;Corrected printer I/O
;   V6.6a  10/1/2011   ;Changed patch to E000:2000H along with JMPF
;   V6.7   10/1/2011   ;Added CGA video board routines for MS-DOS outputs (Major changes to Console IO routine, INT 10H)
;   V6.7b  10/1/2011   ;Added AT-BIOS INT 10, Write String function

```

```

;   V6.7c  10/1/2011    ;IO Byte switch to redirect CRT Output to CGA/VGA Display
;   V6.8    10/4/2011    ;"O" CMD to relocate total Monitor to RAM at E000:8000H
;   V6.9    10/5/2011    ;Initilize CGA RAM memory variables in @449H. Cleanup SETUP_IBM_BIOS:
;   V7.0    10/9/2011    ;General cleanup
;   V7.1    10/15/2011   ;Added Print Screen function to IBM BIOS
;   V7.2    10/18/2011   ;Added Ctl+Alt+Del, Pause etc to keyboard functions.
;   V7.3    11/22/2011   ;Added menu options to set time & date with CMOS-RTC on S100Computers/N8VEM MSDOS Support board
;   V7.4    11/23/2011   ;Switched over to using 8253 timer tick exactly as IBM-AT does.
;   V7.5    11/23/2011   ;Improve 8259A/keyboard test
;   V7.51   11/26/2011   ;Clear 8259A on power up
;   V7.6    12/27/2011   ;Fixed lack of CR/LF for "S" CMD.  Changed "P" CMD to display RAM words ("D" displays RAM Bytes)
;   V7.7    2/19/2011    ;Allow IOBYTE to decide if extra ROMS are to be initilized
;   V7.8    3/10/2012    ;IOBYTE = 0BFH, forces diagnostic test at FFFF0H
;   V7.9    3/27/2012    ;Corrected IDE Board hanging with diagnostic if only one or no drives present, Also numerous other small
bugs
;   V8.0    3/31/2012    ;Corrected "L" command initilizing 8259A to not allow ints during initilization
;   V8.1    3/31/2012    ;Corrected Set Time for MS-DOS Board Dallas RTC Chip
;   V8.2    5/4/2012     ;Major Upgrade, Added LAVA-10 Video display output
;   V8.3    7/6/2012     ;IOBYTE debug changes at end.
;   V8.4    7/7/2012     ;Problem finding default MSDOS video output.
;   V8.5    8/13/2012    ;MSDOS not booting up with CGA board corrected
;   V8.6    8/14/2012    ;MSDOS not booting up with 80286 board was not defaulting to Propeller board (CONSOLE_FLAG not using ES:=0)
;   V8.7a   10/21/2012   ;Changed "L" command. Was not working correctly for 8259A test. Byte[CONSOLE_FLAG] errors
;   V8.8a   10/21/2102   ;Change CONSOLE_FLAG to word. This way less chance or random 0,1,or 2 appearing in RAM at startup
;   V9.0    2/6/2013     ;Spliced in many modifications & improvements from RM 80386 monitor. Add Modem routine
;   V9.1    2/8/2013     ;Fixed (80386) JZ,JNZ jumps so they are within range for 8086.  --- REMOVED 80386 CODE ---

```

```

; V9.2  2/15/2013      ;Patch: cleaned, up now starts at 100H in RAM
; V9.3  5/3/2013      ;Patch done for 8086/8088, 100H -> E8000H (Note. Different from 80386 Monitor)
; V9.3A 5/3/2013      ;Treat 8086/80286 and 8088 CPU board ROM locations different. Removed Y command
; V9.4  5/7/2013      ;Improved U command, general cleanup
; V9.5  5/17/2013     ;Simplify checking/initilizing VGA EPROM (VGA_ROM_CHECK:)
; V9.6  5/23/2013     ;Added INT 15H, AH=44H for Cirrus Logic VGA Board initilization
; V9.7  7/10/2013     ;Rearranged & set to 0 low RAM storage variables at ~500H
; V9.8  7/12/2013     ;Add coded at FF065H to jump to VIDEO_IO routines (For EGA Board)
; V9.8a 7/12/2013     ;Corrected low RAM initilization of INT verctor relocations
; V10.0 7/15/2013     ;EGA board working is S100 bus with ISA->S100 converter. Changed equates for 80286 EEPROMS
; V10.1 7/29/2013     ;European Cruise updates
; V10.2 8/31/2013     ;Correct bit display for memory compare error (VERIFY:)
; V10.3 10/23/2013    ;Stop duplicate "ZFDC not present" message upon initilization
; V10.31 6/14/2014    ;Minor change to scroll up routine
; V10.32 7/17/2014    ;Correction of Y command in IDE_SIGNON3 menu
; V10.33 7/25/2014    ;Added capability to switch back to Z80 with Z command utilizing the V2 SMB (port EEH).
; V10.33a 8/8/2014    ;Remove the '.' diagnostic character for CGA screen scroll etc. put in normal space character
;
; To Do:-
; Caps Lock key is showing as ^ on for MS-DOS command line on Propeller & LAVA video boards (Fix on propeller board)
;
;
; Notes...
; This fairly extensive 8086+ monitor consists of 3 main sections. It assumes an 8086 (does not use opcodes of the 80286+)
; Section 1. This is a classical monitor. Display, change RAM/ports etc.

```

```

;      Section 2.   This is a self-contained set of routines run diagnostic tests on the S100Computers/N8VEM IDE board.
;
;      Section 3.   This fairly complex section. It emulates most of the IBM-PC ROM BIOS interrupts (hard & soft) such that MS-DOS
;                   (V4.01)/FreeDOS can be run on the system - without DOS disk modifications.
;
;
;      In the final EEPROM, code will be placed at F8000H for the 8088/8086 Boards (or F0000H for the S100Computers/N8VEM 80286
board).
;
;      This is because the 8086/8088 boards only have capacity for 32K using 28C256 EEPROMS. The 80286 can fit 64K with EEPROMS.
;      (BTW, the 8086/8088 boards can accomidate 64K using UV-EPROMS (27C256's) and so can reside at F0000H if required).
;
;
;      In all cases the stack is at DFFFCH & IDE Board RAM buffers/BP are at DE000H.
;      For debugging/testing this monitor normally will reside in RAM at E8000H
;
;      Note, the stack is still at DFFFCH & IDE RAM buffer/BP at DE000H.
;
;
;      This monitor needs a valid stack in RAM. It first checks if there is valid RAM in high memory below the ROM
;      (The EEROM is usually at F000:0000H). If so it will set the SS to D000H and the SP to FFFCH. This puts it out of the way
;      of everything in low RAM. If it does not detect RAM there, it will search for a valid segment at top of RAM downwards and
;      put the stack there.
;
;
;      That is the only RAM the main PM monitor needs. However the IDE drive diagnostic routines require much more (sector buffers
etc.)
;
;      For these I have set the SS:BP to DE000H. We use SS:BP throughout to access that RAM for the IDE Board diagnostic section.
;
;
;      Remember also, that RAM at 0-3FFH is initialized to trap all 8086 interrupts. If you want this, activate the
;      SETUP_INT_TABLE routine at the start of this code. Otherwise they are not used/activated unless you use the "L" and "I"
commands.
;
;
;      Most monitor commands are modeled after the old TDL/Zapple/Z80 commands. Because we are now dealing with potentially up

```

```

;   to 1MG of RAM for many commands, the start, end RAM locations etc. can take up to 5 digits.
;   However the actual span/range for any command is limited to 64K.
;
;   The following example fills RAM with 76H from 1A000H to 21234H.
;   F1A000,21234,76
;
;   Of course for the lowest 64K of RAM the "normal" 4,3,2 or 1 byte formats can be used
;   F123,456,76
;
;   Note because of the 64K range limitation the following will give an error
;   F1A000,31234,76   or F1A000,2A001,76
;
;   In general hitting the ESC key will abort any long display/command function.
;   In all cases, to accept data, finish the entry with a CR.
;   So if the display says "xxxxH" you enter up to 4 hex digits than a CR (No "H" is required).
;
;   The Console OUT routines CAN be different, The "Normal" monitor and IDE diagnostic sections ALWAYS go through the
;   Propeller driven (or any serial type) Console out routine (CO: etc). This by default is also the case
;   when the IBM-BIOS sections are activated. However if the Console output is redirected to the CGA/VGA board (INT 10H etc),
;   then CO: is not used. Instead CO goes to the IBM BIOS video board output. This is controlled by the [CONSOLE_FLAG] byte in low
RAM
;   or by switches on the IOBYTE port (see below).
;
;   Likewise, the Console IN routines CAN be different. The "Normal" monitor and IDE diagnostic sections ALWAYS go through the
;   Propeller driven (or any serial type) Console IN routine (CI:, CICO, etc).
;   However when the IBM BIOS section is active (X Menu command, booting up MSDOS etc.), Keyboard input is ALWAYS interrupt
;   driven, and requires the 8259A PIC etc. It uses the input from the propeller board, but each key press interrupt
;   places the data in the IBM-PC style RAM buffer for later retrieval.

```

```

; If the 8259A interrupt functions are not working this section (MS-DOS CI), will hang! You can use the 8259A diagnostics command
; (Main menu, "L" command) to debug this section beforehand.
;
; For Old (< V5.2) Versions Assembled with Digital Research's ASM86, see this section in those files.
;
; For all New Versions (> V6.0) Assemble to a binary file with the excellent/free MSDOS/FreeDOS, NASM.EXE Assembler
; NASM -f bin 8086.A86 -o 8086.bin -l 8086.lst
;
; This will make an 8 bit format .bin file
;
; There are a number of ways to test/run this monitor. Until you actually have this monitor in EEPROM in high memory, you
; can run it low RAM (say 4800H). Until you have such a working version you must have your 8086 after reset jump to this
; test monitor in low RAM at 4800H. Remember in this case you will may have to hand code the stack to a different
segment/location.
;
; To move it across to your CPM80 disk file system (Telnet/Modem/serial connection, whatever. I use PCLOAD.COM).
; PCLOAD.COM can be downloaded from the www.S100Computers.com web site. You can use this program to place the code
; anywhere in the lower 64K space (or if you have the S100Computers/N8VEM Z80 board, anywhere in the 1M 8086 address space).
;
; After switching to your 8086 Board (IN port EDH), have the 8086 jump to there with
; @FFFF0H:- EA 00 xx 00 00 Where xx is a lower 64K, 1K boundary (eg. EA 00 48 00 00).
;
; Later when you have the EPROM monitor working, for debugging, you can use the 8086 monitor "W" command to load
; a .bin file sent from a PC (using the XMODEM format) to anywhere in RAM, normally at E8000H
; and then use the "G" command to jump to the start of the test monitor.

```

```

;

;   Remember for debugging, the code must start at E8000H. The first 8000H bytes in the code will be 0's for the 8086/8088 boards
;   so with the "W" command set the load address to E0000H to have the actual monitor reside at E8000H. Again this is because the
8086 & 8088
;   boards only accommodate 32K of EEPROM. As mentioned above, the code can begin at E0000H for both these boards with 27C256's or
on the 80286 board.

;

;   As soon as you get things going, burn a EEPROM version that resides at F8000H.

;   After switching to your 8086/8088 Board (IN port EDH), the Monitor should immediately come up.

;

;   From then on, it is best to keep RAM test versions up in the 8086 high RAM. That way you can test MSDOS etc.

;   I use the location E8000H. You can use the Monitor "W" and "G" Commands

;   This saves keystrokes, for the many times you do this!

;

;   To burn two 28C256 EEPROM's with a Wellon VP280 or V290 Programmer...

;   Load .BIN file. Select Even bytes (1st of 2) for one ROM and "From File HEX address" and "Buffer Address"

;   leave 0000 in the dialog boxes, do not change "File Size (HEX) either".

;   Repeat for ODD addresses. In each case the Edit Box code should appear from 4000H-7FFFH if the ROM is read back.

;

;   To burn a single 28C256 EEPROM (for the 8088 CPU board) with a Wellon VP280/VP290 Programmer...

;   From "File HEX address" = 8000, do not change "Buffer Address" or "File Size (HEX)"

;   (The Edit BOX the code should appear at 0000H-7FFFH).

;

;   To burn two (Even & Odd Bytes) 27128 EEPROM's with a Wellon VP280 Programmer...

;   From "File HEX address" = 8000, do not change "Buffer Address" or "File Size (HEX)"

;   (The Edit BOX the code should appear at 0000H-3FFFH).

```



```

;
;   One final note, the monitor has now got quite large with only a few bytes to spare to fit in
;   the F8000H - FFFFFH range.  If you add more code you may need to remove other code or shorten the text messages
;   If you are using the 80286 board you have can use the full 27C256 EEPROMS and have the code start at F0000H.
;   (Without patching the 8086 board jumpers you cannot do this with this board).
;   Alternatively you can use the 80386 Monitor.
;

```

```

SCROLL      EQU    01H          ; Set scrool direction UP.
BELL        EQU    07H
SPACE       EQU    20H
BS          EQU    08H
TAB         EQU    09H          ; TAB ACROSS (8 SPACES FOR SD-BOARD)
CR          EQU    0DH
LF          EQU    0AH
FF          EQU    0CH
QUIT        EQU    11H          ; Turns off any screen enhancements (flashing, underline etc).
ESC         EQU    1BH
DELETE      EQU    7FH
BACKS       EQU    08H
CLEAR       EQU    1AH          ; TO CLEAR SCREEN
SOH         EQU    1          ; For Modem etc.
EOT         EQU    4
ACK         EQU    6
NAK         EQU    15H

```

```
MONITOR_ROM EQU TRUE ;TRUE = put monitor reset code at F800:FFF0H, FALSE = skip
;Set to FALSE for debugging code in RAM
```

```
CPU_80286      EQU      FALSE      ;True if an 80286,  FALSE if 8086 or 8088 CPU

;The 80286 board allows easily allows an address range for F0000H-FFFFFH.

;The 8088/8086 boards normally (28C256's) allow a range of F8000H-FFFFFH

;The the IBM-PC character table section is not available for the 8086/8088 CPU's

;because there is not enough room for 28C256's. (BTW, there is for UV 27256 EPROMs).

;It is seldome used anyway, It uses a lot of space (see CRT CHAR GEN:)
```

```
CPU 8086      EQU      FALSE      ;True if an 8086,  FALSE if 8088  (Just used in signon message)
```

```
;Propeller Console IO S-100 board or SD SYSTEMS VIDIO BOARD FOR CONSOLE I/O(<---These must configured for your hardware)
```

```

KEYSTAT      EQU    0H

KEYIN        EQU    01H      ;Console input port. Normally the Propeller Driven S-100 Console-IO Board

KEYOUT       EQU    01H      ;Console output port. Normally the Propeller Driven S-100 Console-IO Board

```

```

;----- THIS IS MY PORT TO OUTPUT DATA TO HP 4050T LASAR PRINTER  (IMSAI 8PIO Board)

```

```

PRINTER_STATUS EQU    5      ;IN, HP PARRELL PORT

PRINTER_OUT  EQU    5      ;OUT

PRINTER_STROBE EQU    4      ;OUT

DIAG_LEDS    EQU    5      ;OUT (Will use this port initially for diagnostic LED display)

```

```

;----- S100Computers Serial I/O BOARD PORT ASSIGNMENTS  (A0-AC)

```

```

BCTL        EQU    0A0H      ;CHANNEL B CONTROL PORT ASSIGNMENTS OF THE ZILOG SCC CHIP

ACTL        EQU    0A1H      ;CHANNEL A CONTROL

BDTA        EQU    0A2H      ;CHANNEL B DATA

ADTA        EQU    0A3H      ;CHANNEL A DATA

;

PortA_8255  EQU    0A8H      ;A port of 8255 ;<--- Adjust as necessary

PortB_8255  EQU    0A9H      ;B port of 8255

PortC_8255  EQU    0AAH      ;C Port of 8255

PortCtrl_8255 EQU    0ABH      ;8255 configuration port

```

```

AinBout8255cfg      EQU    10011000b      ;Set 8255 ports:- A input, B output,
                                     ;C(bits 0-3) output, (bits 4-7)input

USB_DATA            EQU    0ACH            ;PORT ASSIGNEMENT FOR DLP-USB Controller chip
USB_STATUS          EQU    0AAH            ;Status port for USB port (Port C of 8255, bits 6,7)

USB_RXE             EQU    80H             ;If Bit 7 = 0, data available to recieve by S-100 Computer
USB_TXE             EQU    40H             ;If Bit 6 = 0 data CAN be written for transmission to PC

BASE_PORT           EQU    0A1H

MODEM_CTL_PORT      EQU    BASE_PORT       ;A1H
MODEM_SEND_MASK     EQU    4
SEND_READY          EQU    4              ;VALUE WHEN READY
MODEM_RECV_MASK     EQU    1
RCV_READY           EQU    1              ;BIT ON WHEN READY
MODEM_DATA_PORT     EQU    BASE_PORT+2     ;A3H

RECV_SECT_NO        EQU    100H            ;BP Offset for Recieved Sector Number
SECTNO              EQU    101H            ;BP Offset for CURRENT SECTOR NUMBER
ERRCT               EQU    102H            ;BP Offset for ERROR COUNT
MODEM_ERR_LIMIT     EQU    8              ;Max number of Modem serial port re-reads aborting
MODEM_RTS_DELAY     EQU    1              ;Time to check Modem RTS line (See XMODEM_LOAD & P_XMODEM_LOAD). Not critical.

```

```

;----- S100Computers SMB Board -----

IOBYTE      EQU      0EFH      ;IOBYTE Port on S100Computers SMB Board.

;IOBYTE = SENSE SWITCHES AT PORT 0EFH
;
; BIT MAP OF PORT 0EFH:- X X X X  X X X X      (11111111=NORMAL CONFIG)
;
;           | | | |  | | | |...For Z80 Monitor,      0=CONSOLE DATA TO PRINTER ALSO
;
;           | | | |  | | | |....For 8086 Monitor,    0=Force MSDOS Consol output to CGA/VGA Board instead of Propeller board
;
;           | | | |  | | | |.....For 8086 Monitor,   0=Do not initilize extra ROMS
;
display     | | | |  | | | |.....For 8086 Monitor,   0=Debug data for all MSDOS 10H & 15H INT's to Serial Port and external
;
;           | | | | |.....Unused
;
;           | | | |.....For Z80 Monitor,             0=ALL Consol I/O via ACIA Serial port on S100Computers Serial-IO Board
;
;           | | | |.....For CPM3,                   0=Force reformat of Memory disk upon any CPM3 cold re-boot
;
;           |
;
;           |.....For CPM3,                          0=Write protect Memory disk for CPM3
;
;           For 8086 Monitor,                        0=Prevent doing a JMPF to 500H after 8086 reset (to CPM86 boot)
;
;                                           Normally a test is made to see if the CPM86 Boot is already in RAM at 500H
;
;                                           If it is, a 8086 reset will bypass the monitor and go directly there.
;
;                                           (see Init: in 8086 Monitor)
;
;
; Note if IOBYTE = 00xxxxxx,                This will force the hardware diagnostic test after reset.
;
;                                           (See code at FFFF0H in this 8086 monitor)
;
;
;

```

```
;----- S100Computers MSDOS BOARD PORT ASSIGNMENTS
```

```
NS_EOI      equ    20h          ;Non specific end of interrupt command

MASTER_PIC_PORT    equ    20h          ;Hardware port the 8259A is assigned (two ports 20H & 21H)


MasterICW1    equ    00010111B      ;EDGE triggered, 4 bytes, single Master, ICW4 needed
MasterICW2    equ    8H              ;Base address for 8259A Int Table (IBM-PC uses 8X4 = 20H)
MasterICW3    equ    0H              ;No slave
MasterICW4    equ    00000011B      ;No special mode, non buffer, Auto EOI, 8086.  ;<<<<,


CMOS_PORT     EQU    70H            ;Base Port for CMOS Chip
CMOS_VALID    EQU    0DH            ;To check DS12887 CMOS chip is present and OK (Note AT-BIOS uses 0EH)
CMOS_REGA     EQU    0AH            ;CMOS REGISTER A


TIMER         EQU    40H            ;Base port of 8254
TIM_CTL              EQU    43H

COUNTS_SEC   EQU    18
COUNTS_MIN   EQU    1092
COUNTS_HOUR  EQU    07H           ;Seems this value is used with AT/CMOS chip (was 65543 on PC)


UPDATE_TIMER  EQU    80H


CMOS_SECONDS  EQU    0H            ;RAM offsets for CMOS Registers
```

```
CMOS_MINUTES EQU    2H
```

```
CMOS_HOURS   EQU    4H
```

```
;----- S100Computers IDE BOARD PORT ASSIGNMENTS  (30-34H)
```

```
;Ports for 8255 chip. Change these to specify where the 8255 is addressed,
```

```
;and which of the 8255's ports are connected to which IDE signals.
```

```
;The first three control which 8255 ports have the IDE control signals,
```

```
;upper and lower data bytes. The forth one is for mode setting for the
```

```
;8255 to configure its ports, which must correspond to the way that
```

```
;the first three lines define which ports are connected.
```

```
IDEportA     EQU    030H          ;lower 8 bits of IDE interface
```

```
IDEportB     EQU    031H          ;upper 8 bits of IDE interface
```

```
IDEportC     EQU    032H          ;control lines for IDE interface
```

```
IDECtrlPort  EQU    033H          ;8255 configuration port
```

```
IDEDrivePort EQU    034H          ;To select the 1st or 2nd CF card/drive
```

```
IDE_Reset_Delay EQU    020H          ;Time delay for reset/initilization (~66 uS, with 8MHz 8086, 1 I/O wait state)
```

```
READcfg8255  EQU    10010010b      ;Set 8255 IDEportC out, IDEportA/B input
```

```
WRITEcfg8255 EQU    10000000b      ;Set all three 8255 ports output
```

```
;IDE control lines for use with IDEportC.
```

```

IDEa0line    EQU    01H        ;direct from 8255 to IDE interface
IDEa1line    EQU    02H        ;direct from 8255 to IDE interface
IDEa2line    EQU    04H        ;direct from 8255 to IDE interface
IDEcs0line   EQU    08H        ;inverter between 8255 and IDE interface
IDEcs1line   EQU    10H        ;inverter between 8255 and IDE interface
IDEwrline    EQU    20H        ;inverter between 8255 and IDE interface
IDERdline    EQU    40H        ;inverter between 8255 and IDE interface
IDERstline   EQU    80H        ;inverter between 8255 and IDE interface

```

```
;
```

```
;Symbolic constants for the IDE Drive registers, this makes the
```

```
;code more readable than always specifying the address pins
```

```

REGdata      EQU    IDEcs0line
REGerr       EQU    IDEcs0line + IDEa0line
REGsecCnt    EQU    IDEcs0line + IDEa1line
REGsector    EQU    IDEcs0line + IDEa1line + IDEa0line
REGcylinderLSB EQU    IDEcs0line + IDEa2line
REGcylinderMSB EQU    IDEcs0line + IDEa2line + IDEa0line
REGshd       EQU    IDEcs0line + IDEa2line + IDEa1line      ; (0EH)
REGcommand   EQU    IDEcs0line + IDEa2line + IDEa1line + IDEa0line ; (0FH)
REGstatus    EQU    IDEcs0line + IDEa2line + IDEa1line + IDEa0line
REGcontrol   EQU    IDEcs1line + IDEa2line + IDEa1line
REGastatus   EQU    IDEcs1line + IDEa2line + IDEa1line + IDEa0line

```



```
;IDE Command Constants.  These should never change.
```

```
COMMANDrecal EQU    10H
COMMANDread  EQU    20H
COMMANDwrite EQU    30H
COMMANDinit  EQU    91H
COMMANDid    EQU    0ECH
COMMANDspindown EQU  0E0H
COMMANDspinup EQU    0E1H
```

```
;
```

```
; IDE Status Register:
```

```
; bit 7: Busy      1=busy, 0=not busy
; bit 6: Ready 1=ready for command, 0=not ready yet
; bit 5: DF 1=fault occurred on the IDE drive
; bit 4: DSC1=seek complete
; bit 3: DRQ1=data request ready, 0=not ready to xfer yet
; bit 2: CORR      1=correctable error occurred
; bit 1: IDXvendor specific
; bit 0: ERR1=error occurred
```

```
MAXSEC      EQU    3DH          ;Sectors per track for CF my Memory drive, Kingston CF 8G. (CPM format, 0-3CH)
                                   ;translates to LBA format of 1 to 3D sectors, for a total of 61 sectors/track.
                                   ;This CF card actually has 3F sectors/track. Will use 3D for my CPM86 system because
                                   ;my Seagate drive has 3D sectors/track. Don't want different CPM86.SYS files around
                                   ;so this program will also work with a Seagate 6531 IDE drive
```

```

DOS_MAXSEC      EQU      3FH          ;For MS-DOS BIOS Setting "Hard Disk" to Custom type (CF Card, 63 Sectors/track)
DOS_MAXHEADS    EQU      10H          ;16 head(s)
DOS_MAXCYL_L    EQU      0FFH        ;Low Byte maximum cylinder (sent via INT 13H's in CH)
DOS_MAXCYL      EQU      1024        ;Max cylinders
DOS_MAXSEC_CYL  EQU      0FFH        ;3FH, maximum sector number (bits 5-0)+ two Cyl High Bits (Sectors numbered 1....x)

```

```

;-----S100Computers PORTS FOR FOR Z80/WD2793 FDC Board

```

```

S100DATAA      EQU      10H          ;IN, S100 Data port to GET data to from FDC Board
S100DATAB      EQU      10H          ;OUT, S100 Data port to SEND data to FDC Board
S100STATUSA    EQU      11H          ;Status port for A
S100STATUSB    EQU      11H          ;Status port for B
RESETZFDCPORT  EQU      13H          ;Port to reset ZFDC Z80 CPU.

STATUSDELAY    EQU      20           ;Time-out for waiting for ZFDC Board handshake signal (Now, ~0.5 seconds @ 8MHz 8086)
SECTOR_TIMEOUT EQU      400H         ;Value for sector R/W status check countdown (For 6-8MHz 8086, not critical)

ZFDCUNINITIALIZED EQU      0FFH      ;If ZFDC is not yet initilized
ZFDCNOTWORKING  EQU      0FEH        ;If ZFDC is not working
ZFDCNOTPRESENT  EQU      0FDH        ;If ZFDC board is absent
ZFDCINITIALIZED EQU      000H        ;If ZFDC is initilized OK

STD8IBM        EQU      1           ;ZFDC Board Format table # for IBM 8" SDSS Disk

```

```

MSDOS2      EQU    13H      ;Disk format type # for ZFDC board  (MS-DOS V2.0 Disk, 512 X 9 Sec/Track)

IBM144      EQU    15H      ;Disk format type # for 1.4M DDDS, 18 X 512 Byte Sectors, 80 Tracks


CMD_SET_FORMAT      EQU    4H      ;This will select a specified drive and assign a disk format table to that drive
CMD_SET_DRIVE EQU    5H      ;This will select a specified drive (0,1,2,3)
CMD_SET_TRACK EQU    7H      ;This will set head request to a specified track
CMD_SET_SIDE EQU    8H      ;This will set side request to a specified side
CMD_SET_SECTOR      EQU    9H      ;This will set sector request to a specified sector
CMD_SET_HOME EQU    0AH      ;This will set head request to Track 0 of CURRENT drive
CMD_SEEK_TRACK      EQU    0EH      ;Seek to track to (IY+DRIVETRACK) with the track verify bit set on CURRENT drive/format
CMD_FORMAT_TRACK EQU    16H      ;Format the floppy disk in the of the CURRENT drive using the current format assigned to that
disk
CMD_HANDSHAKE EQU    21H      ;Handshake command only sent during board initialization/testing


;These new commands are required for R/W MSDOS Double sided disks

CMD_DOS_RD_MULTI_SEC      EQU 2BH      ;MS-DOS, Read data from multiple sectors starting at the CURRENT sector.
CMD_DOS_WR_MULTI_SEC      EQU 2CH      ;MS-DOS, Write data to multiple sectors starting at the CURRENT sector.
CMD_GET_SIDE      EQU 2DH      ;Get the current selected side of the current selected drive
CMD_DOS_SET_SECTOR      EQU 2EH      ;MS-DOS, Set current sector for the next sec R/W

```

;Possible ERROR codes returned from the ZFDC Board:-

;These will be translated into ASCII strings in the error reporting function.

;See the ZFDC code for a complete set of possible error coded returned byt the ZFDC Board

```

NO_ERRORS_FLAG      EQU      00H          ;No Errors flag for previous cmd, sent back to S-100 BIOS
CONFIRM_FORMAT       EQU      32H          ;Confirm disk format cmd request
DISK_WP_ERR         EQU      31H          ;Sector write error, Disk is write protected
ABORT_FLAG          EQU      3AH          ;Special error flag to signify the user aborted a command

ZFDC_ABSENT         EQU      3BH          ;If ZFDC Board is absent
ZFDC_INIT_ERROR      EQU      3CH          ;If ZFDC initialization error

TIMEOUT_ERROR        EQU      3DH          ;Error flag to signify the previous command timed out
CMD_RANGE_ERROR      EQU      3EH          ;CMD out of range.

MAX_ERRORS           EQU      3FH          ;0 to 3FH errors only

;Meanings for disk status (as returned by IBM BIOS ROM)

seekerr              equ      40h          ;seek failed
hdwerr               equ      20h          ;controller chip failed
crcerr               equ      10h          ;crc error
dmaerr               equ      09h          ;DMA across 64k boundary
wpterr               equ      03h          ;write protected disk
rnferr               equ      04h          ;sector not found
timerr               equ      80h          ;Floppy time out error
cmderr               equ      01h          ;Floppy bad command for controller

msize                equ      280H          ;Total RAM memory size, (640K)
romdat               equ      0h          ;Data area for ROM usage (DS will be set to 0H for data at 400H....)

```

```
;----- CGA/VGA/XVGA Video board equates -----
```

```
c6845port      Equ    3d0h          ;base port for Lomas/CGA colour board
bw6845port      Equ    3b0h          ;base port for b/w card
Index_Reg_Count Equ    16            ;Count of 6845 Index registers
```

```
;----- LAVA-10 Video board equates -----
```

```
LavaStatus      EQU    090H          ;Status Port
LavaData         EQU    091H          ;Data port
```

```
;LAVA Commands:-
```

```
COPY$MEMORY      EQU    010H
WRITE$MEMORY      EQU    020H
READ$MEMORY       EQU    030H
DRAW$TEXT         EQU    040H
READ$CSR          EQU    036H
WRITE$CSR         EQU    022H
```

```
L_CRT_WIDTH      EQU    800          ;Pixels across per line
L_CRT_HEIGHT     EQU    600          ;Pixels (16 per line)
L_BELOW_SCREEN   EQU    601          ;Area of LAVA screen RAM not visible (Use as a clear buffer area, EOL etc)
L_CHARS_PER_LINE EQU          99      ;99X8 = 792
```

```

L_SCREEN_LINES    EQU    37            ;37X16 = 592
L_CHAR_WIDTH EQU    8                ;Character pixel width
L_CHAR_HEIGHT EQU    16                ;Character pixel height

```

```

L_WHITE_COLOR EQU    0FFFFH
L_BLACK_COLOR EQU    00000H
L_BLUE_COLOR  EQU    00F0FH
L_GREEN_COLOR EQU    008F0H

```

;-----Other Hardware Equates -----

```

HOLD_STATE    EQU    80H            ;Set Keyboard flag to indicate a Pause is required
NO_HOLD_STATE EQU    7FH            ;To clear the above flag

```

```

SW86          EQU    0EDH            ;INPUT FROM THIS PORT SWITCHES THE 8086/80286 BACK to the Z80 in hardware
SW86_TM        EQU    0EEH            ;Output 00H to this port to switch back to Z80 Hardware (on SMB V2 only).
SW68K          EQU    0ECH            ;SWITCHES TO THE 68K TO MASTER MODE FROM THE Z80 CPU, (not used here).

```

```

STACK_SEG     EQU    0D000H          ;Normally (Stack D000:FFFCH)
STACK_POINTER EQU    0FFFFH          ;With 1M RAM Stack will normally be at D000:FFFC
BASE_POINTER  EQU    0E000H          ;Default BP at D000:E000H

```

```

;===== Start of BIOS code segment =====

        CPU 8086          ;No 80286/386 opcodes
        [BITS 16]

SECTION .text

        org      0H

        %if      CPU_8088
TIMES 8000H DB      0H          ;To have code in EEPROMS start at F8000H for 8088/8086 board
        %endif                ;(Note for the the 80286 board the Monitor can start at F0000H with 28C256 EEPROMs)

        %if      CPU_8086
TIMES 8000H DB      0H          ;To have code in EEPROMS start at F8000H for 8088/8086 board
        %endif                ;(Note for the the 80286 board the Monitor can start at F0000H with 28C256 EEPROMs)

        ;All addresses will be relative to this location F/E8000H

        ;For 8086 debugging/testing this monitor will reside in RAM at E000:8000H with
        ;the stack at D000:FFFCH.

BEGIN: jmp      INIT          ;Reset all registers, initilize hardware
        jmp      WARM_INIT     ;warm start

```

```

    jmp    CI                ;console input
    jmp    RI                ;reader output
    jmp    CO                ;console output (Character in CL)
    jmp    POO              ;punch output
    jmp    LIST_OUT         ;printer output (Character in CL)
    jmp    CSTS             ;consol status
    jmp    CICO             ;console in with echo
    jmp    LIST_STATUS      ;printer status

```

Test_code:

```

    MOV    al,81h
zzz:  out    80h,al
    jmp    zzz

```

```

INIT:  cld                ;Set direction up. Through this monitor this is the default direction
        cli                ;Disabel interrupts initially

```

```

MOV    AL,00000000B        ;ALL LED's ON, for VISUAL DIAGNOSTIC we are alive
OUT    DIAG_LEDS,AL        ;LED's will go off one at a time

```

```

    IN     AL,IOBYTE        ;If bit 8 of Port EFH is 0, Then force jump to this Monitor (Note, If 0, RAM disk with CPM3 will be
invalid)
    AND    AL,80H           ;If bit 1 is 1 then see if CPM86/MSDOS is present in RAM at 0000:0500H
    JZ     ToMonitor        ;IF so, then jump to that loction. If 0xxxxxx, skip this check

```



```

MOV     BX,500H                ;Normally my CPM86.COM (or MSDOS.COM) program will have 90H,90H in RAM at 500H.
                                     ;So that for CPM+ one can type "CPM86" and boot the system up directly from a Z80's CPM+
                                     ;This is just a flag I use as a check for the 8086 monitor not to jump to 500H in RAM
                                     ;if nothing is there.

MOV     AX,0                   ;Check this value is here. If so, chances are we have CPM86 or MSDOS loaded (but unitilized) in RAM
MOV     DS,AX                  ;If not then skip to this monitor
CMP     word[BX],9090H         ;Was it a reset requiring CPM86.
JNZ     ToMonitor              ;Set pointers for IBM-PC BIOS interrupt vectors in low RAM

                                     ;NOTE: The only problem with the above is if you reset your system and 90H,90H is still
                                     ;in RAM at 500H, switching the 8086 back in (Z80 "O" CMD) will probably crash as the monitor
                                     ;will transfer control down to an already initilized CPM86 system.
                                     ;The solution is simply first erase the 90H,90H in RAM @ 500H or use the above IOBYTE switch.
                                     ;If you don't like the above option, just comment out this code and jump directly to "ToMonitor"

CPM86Boot:                      ;If 90H,90H at 500H in RAM
    JMP     word 0000H:500H     ;Far Jump to 500H in RAM (where CPM86 resides)

ToMonitor:                     ;If not, then jump to this 8086 Monitor in ROM
    cld                        ;Set direction up "UP" troughout this monitor, this is the default direction
    cli                        ;Disabel interrupts

MOV     AL,10000000B           ;1st LED off, for VISUAL DIAGNOSTIC we are alive
OUT     DIAG_LEDS,AL           ;LED's will go off one at a time as we progress through initilization

```

```

;We will now set up a valid stack. Normally there will be 1MG of S-100 RAM
;in the system so there will be RAM below this EEPROM at D000:FFFCH
;If so, we will place the stack below the EPROM.
;If however there is less memory we will find the highest RAM and place
;the stack at the top of available RAM

```

```

mov     ax,cs           ;Note cs will be F000H (or E000H)
mov     ds,ax           ;DS will also be CS:F000H (or E000H)
mov     es,ax           ;As will ES
mov     ax,STACK_SEG    ;Will start with D000H. This leaves 64K at E000:0H for debugging/test versions
mov     ss,ax           ;For now, SS also set to D000H
mov     sp,STACK_POINTER ;Initial SP=D000:FFFCH
mov     bp,BASE_POINTER ;BP for IDE RAM variables (will normally be SS:[BP] or D000:E000H)

```

```

mov     BX,STACK_POINTER ;Normally D000:FFFCH

```

```

TOP_OF_RAM:           ;Check if RAM there, if not find TOS

```

```

MOV     AX,[SS:BX]
NOT     AX
MOV     [SS:BX],AX
CMP     [SS:BX],AX     ;Is there real RAM there.
JNZ     NO_RAM         ;If no RAM then search for lower memory

```

```

MOV     SP,BX

```

```

JMP     DoneStack

```

```

NO_RAM:      MOV     AX,SS                ;Try 64K lower....

             SUB     AX,1000H

             MOV     SS,AX

             OR      AX,AX

             JNZ     TOP_OF_RAM

             MOV     SP,4FCH              ;Special case if <= 64K RAM

                                         ;Point to a RAM area (0000:4FC), assume we have at least this ammount.

                                         ;In this case BP will be at 0000:E000H. If no RAM there we are out of luck

DoneStack:

             MOV     AL,11000000B         ;2nd LED off, VISUAL DIAGNOSTIC for Stack done

             OUT     DIAG_LEDS,AL

             mov     bx,SIGNON            ;Signon notice

             call    PRINT_STRING         ;Note up to now stack was not used.

             MOV     AL,0FFH              ;Clear Printer strobe, comes up 0 on a reset

             OUT     PRINTER_STROBE,AL

             MOV     AL,11100000B         ;3rd LED off, VISUAL DIAGNOSTIC

             OUT     DIAG_LEDS,AL

                                         ;Initilize the 8259A PIC Controller (This seems to be required to prevent the 8086

             mov     al,MasterICW1        ;from "locking up" after a power on).

```

```

out    MASTER_PIC_PORT,al
mov    al,MasterICW2      ;Ints start at 20H in RAM
out    MASTER_PIC_PORT+1,al
mov    al,MasterICW4      ;No slaves above, so 8259A does not expect ICW3
out    MASTER_PIC_PORT+1,al

mov    al,11111111b       ;Allow No Ints on 8259A for now
out    MASTER_PIC_PORT+1,al

MOV    AL,11110000B       ;4th LED off, VISUAL DIAGNOSTIC
OUT    DIAG_LEDS,AL

mov    bx,MSG
call   SPEAK_STRING       ;Speak out signon the message

MOV    AL,11111000B       ;5 LED's off if speech sent
OUT    DIAG_LEDS,AL

mov    bx,SHOWSTACK       ;Show current stack position
call   PRINT_STRING
MOV    AX,SS
CALL   AX_HEXOUT
MOV    CL,':'
CALL   CO
MOV    AX,SP

```

```

CALL    AX_HEXOUT
CALL    CRLF

;      CALL    SETUP_INT_TABLE          ;Setup RM default INT jump table in RAM 0-3FFH

mov     al,01111111b          ;Send 7FH to 8259A and check it is there
out     MASTER_PIC_PORT+1,al
in      AL,MASTER_PIC_PORT+1
CMP     AL,01111111b          ;Should get same value back if 8259A is present
JZ      PIC_OK

mov     bx,NO_8259A_MSG        ;Send 8259A not found message
call    PRINT_STRING

PIC_OK:  mov     al,11111111b      ;Allow No Ints on 8259A for now
out     MASTER_PIC_PORT+1,al

MOV     AL,11111100B          ;6 LED's off if INT Jump table setup
OUT     DIAG_LEDS,AL

IN      AL,KEYIN              ;Flush keyboard

MOV     AL,11111110B          ;7 LED's off, if ALL initialization is done.
OUT     DIAG_LEDS,AL

```

WARM_INIT:

```

    cld                ;Set direction up
    cli                ;Disabel interrupts
    mov  ax,cs          ;Note cs always will be F000H  (or E000H when testing)
    mov  ds,ax          ;DS & ES will  be set to F000H as default values within this monitor
    mov  es,ax

```

MAINLOOP:

```

    mov  bx,CLEANUP      ;Clear line and '>'
    call PRINT_STRING

    call CICO            ;Get a command from Console
    mov  ah,0
    cmp  al,'A'
    jnb  WARM_INIT       ;must be A to Z
    cmp  al,'Z'
    jg   WARM_INIT
    sub  al,'A'           ;calculate offset
    shl  al,1            ;X 2
    add  ax,ctable
    mov  bx,ax
    mov  ax,[CS:BX]       ;Get location of routine
    CALL AX              ;<-----This is the Main Monitor CMD call
    jmp  WARM_INIT       ;finished

```

```
;***** Basic Monitor Commands *****
```

```
;----- PRINT MENU ON CRT -----
```

```
KCMD:  MOV     BX,MAIN_MENU
        CALL    PRINT_STRING
        RET
```

```
;----- MAP the 1MG Addresss space -----
```

```
MAP:   call    CRLF                ;Display complete memory map with R=ram, P=PROM and "." empty space
        mov     ax,0
        mov     ds,ax              ;Must start in first segment in DS:
        mov     dl,64              ;character count
        mov     dh,4               ;segment counter(4 lines per segment)
        mov     SI,ax              ;need to reset bx (ds = 0 already)
        call    SHOW_ADDRESS_DS    ;start with address, Send to console the address DS+SI

map1:  mov     ax,[SI]              ;remember ds is assumed
        not     ax                  ;complement data
        mov     [SI],ax
```

```

        cmp     ax,[SI]                ;did it change
        jne     not_ram
        not     ax                    ;correct data
        mov     [SI],ax
        mov     cl,'R'
        jmp     nextbk                ;get next block

not_ram: cmp     ax,0                  ;ffff->0 must be rom if not 0
        jne     prom
        mov     cl,','                ;no need to correct data for here
        jmp     nextbk                ;get next block

prom:   mov     cl,'p'
nextbk: call     CO                    ;send the R,P or ","

        ADD     SI,100H                ;check every 100h at a time
        dec     dl                    ;64X1000H across
        jnz     map1                  ;one line of 64K done

        mov     dl,64                 ;reset counter for next line
        dec     dh                    ;segment counter
        jnz     nose1
        mov     ax,ds
        add     ax,1000h
        jc      mapdone
        mov     ds,ax

```



```

        mov     dh,4
noseg: CALL    CRLF_CHECK          ;Print current address at start of each line
        call    SHOW_ADDRESS_DS
        jmp     map1
mapdone:ret

```

```

;-----Fill memory with a constant value. Up to 64K bytes from xxxxxH to xxxxxH-----

```

```

FILL:  CALL    GET5DIGITS          ;Get (up to) 20 bit parameter. 16 bit value (4 digits) to DI.
                                           ;If 5 digits, then the first digit is put in ES (highest nibble)

        PUSH    ES
        PUSH    DI                ;Save start address for now = ES:DI

        CALL    GET5DIGITS

        MOV     SI,DI             ;Put end address in DS:SI
        MOV     AX,ES
        MOV     DS,AX            ;If 5 digits, then the first digit is put in DS

```

```

POP     DI

POP     ES                ;Start=ES:DI  End=DS:SI

CALL    CLENGTH           ;Length cx = (ds:si-es:di)+1, if >64K then err

CALL    GET2DIGITS        ;Fill value to AL (CX unaltered)

                                ;ES:DI = start address, (DS:SI = end address, not used), CX = count, AL = fill value
filoop:  mov     [ES:DI],al    ;Note RAM is es:[di], count in CX
        inc     DI
        CMP     DI,0         ;Check if we are crossing a segment boundry
        JNZ     filoop1
        MOV     AX,ES
        ADD     AX,1000H
        MOV     ES,AX
filoop1:loop filoop        ;Dec CX to 0
        ret

;-----Display memory contents (bytes)

DISPLAY_RAM_BYTES:
        CALL    GET5DIGITS    ;Get (up to) 20 bit parameter. 16 bit value (4 digits) to DI.
                                ;If 5 digits, then the first digit is put in ES (highest nibble)

```

```

PUSH    ES

PUSH    DI                ;Save start address for now.


CALL    GET5DIGITS

MOV     SI,DI             ;Put end address in SI
MOV     AX,ES
MOV     DS,AX             ;If 5 digits, then the first digit is put in DS


POP     DI
POP     ES                ;Start=ES:DI  End=DS:SI


;   AND    DI,0FFF0h      ;even up printout
;   OR     SI,000Fh       ;also nice ending for Ray G.


call    CLENGTH           ;Length cx = (ds:si-es:di)+1, if length > 64K then err


dloop6:    CALL    CRLF_CHECK      ;Note BX,CX is saved, ESC at keyboard will abort
call     SHOW_ADDRESS_ES        ;Send start address


MOV     DL,16             ;First print a line of 16 Hex byte values
PUSH    CX
PUSH    DI
PUSH    ES


dloop1:    mov     al,[es:di]      ;Will increment DI

```

```

call    AL_HEXOUT
call    BLANK
call    Inc_DI_boundry_check      ;Will increase DI
DEC     DL                      ;Have we done 16 bytes yet
jnz     dloop1

                                ;Now print ascii for those 16 bytes
mov     cx,6                    ;first send 6 spaces
call    TABS

MOV     DL,16                    ;16 across again
POP     ES
POP     DI
POP     CX

dloop2:  mov     al,[es:DI]
and     al,7fh
cmp     al,' '                  ;filter out control characters
jnc     dloop3

dloop4:  mov     al,'.'
dloop3:  cmp     al,'~'
jnc     dloop4
PUSH    CX
mov     cl,al
call    CO
POP     CX
loop    dloop5                  ;--CX has total byte count

```

```

        ret

dloop5:    call    Inc_DI_boundry_check

        DEC     DL                ;Have we done 16 bytes yet

        jnz     dloop2

        JMP     dloop6

Inc_DI_boundry_check:                ;Check if we are crossing a segment boundry

        inc     DI                ;If so, inc [ES]

        CMP     DI,0

        JNZ     bounds1

        MOV     AX,ES

        ADD     AX,1000H

        MOV     ES,AX

bounds1:RET

Inc_SI_boundry_check:                ;Check if we are crossing a segment boundry

        inc     SI                ;If so, inc [DS]

        CMP     DI,0

        JNZ     bounds2

        MOV     AX,DS

        ADD     AX,1000H

        MOV     DS,AX

bounds2:RET

```

```
;-----DISPLAY Words Memory -----
```

```
;      This routine forces the CPU to do RAM word reads rather than a byte reads.
;      It is used to test the hardware's ability to do 16 bit reads on odd and even addresses.
;      This is very important. "Normal" byte reads will not show such a hardware problem.
;      A block or RAM/ROM read with the "D" & "P" commands must be identical
```

```
DISPLAY_RAM_WORDS:
```

```
    CALL    GET5DIGITS          ;Get (up to) 20 bit parameter. 16 bit value (4 digits) to DI.
                                ;If 5 digits, then the first digit is put in ES (highest nibble)
```

```
    PUSH    ES
```

```
    PUSH    DI                  ;Save start address for now.
```

```
    CALL    GET5DIGITS
```

```
    MOV     SI,DI               ;Put end address in SI
```

```
    MOV     AX,ES
```

```
    MOV     DS,AX               ;If 5 digits, then the first digit is put in DS
```

```
    POP     DI
```

```
    POP     ES                  ;Start=ES:DI  End=DS:SI
```

```
;    AND     DI,0FFF0h          ;even up printout
```

```
    OR      SI,000Fh            ;also nice ending for Ray Gluck.
```

```

    call    CLENGTH                ;Length cx = (ds:si-es:di)+1, if length > 64K then err
    shr     cx,1                  ;divide by 2 because words

wdloop6:CALL CRLF_CHECK            ;Note BX,CX is saved, ESC at keyboard will abort
    call    SHOW_ADDRESS_ES        ;Send start address

    MOV     DL,8                  ;First print a line of 16/2 Hex byte values
    PUSH    CX
    PUSH    DI
    PUSH    ES

wdloop1:mov  ax,[es:di]            ;Will increment DI
    call    AL_HEXOUT
    call    BLANK
    mov     al,ah
    call    AL_HEXOUT
    call    BLANK
    call    Inc_DI_boundry_check    ;Will increase DI
    call    Inc_DI_boundry_check    ;Will increase DI
    DEC     DL                    ;Have we done 16 bytes yet
    jnz     wdloop1

                                ;Now print ascii for those 16 bytes
    mov     cx,8                  ;first send 6 spaces
    call    TABS

```

```

        MOV     DL,8                ;16/2 across again
        POP     ES
        POP     DI
        POP     CX
wdloop2:mov    ax,[es:DI]
        push    ax
        and     al,7fh
        cmp     al,' '              ;filter out control characters
        jnc     wdloop3
wdloop4:mov    al,'.'
wdloop3:cmp    al,'~'
        jnc     wdloop4
        PUSH    CX
        mov     cl,al
        call    CO
        POP     CX
        pop     ax

        mov     al,ah
        and     al,7fh
        cmp     al,' '              ;filter out control characters
        jnc     wdloop7
wdloop8:mov    al,'.'
wdloop7:cmp    al,'~'

```



```

jnc    wdloop8
PUSH   CX
mov     cl,al
call    CO
POP     CX
loop    wdloop5          ;--CX has total byte count
ret

```

```

wdloop5:call Inc_DI_boundry_check
        call Inc_DI_boundry_check
DEC     DL                ;Have we done 16 bytes yet
jnz     wdloop2
JMP     wdloop6

```

```

;-----MOVE Memory -----

```

```

MOVE:  CALL   GET5DIGITS          ;Get (up to) 20 bit parameter. 16 bit value (4 digits) to DI.
                                           ;If 5 digits, then the first digit is put in ES (highest nibble)
PUSH   ES                      ;Do everything relative to first ES value
PUSH   DI                      ;Save start address for now.

CALL   GET5DIGITS

MOV     SI,DI                  ;Put end address in SI

```

```

MOV     AX,ES
MOV     DS,AX                ;If 5 digits, then the first digit is put in DS

POP     DI
POP     ES                  ;Start=ES:DI  End=DS:SI

call    CLENGTH              ;Length cx = (ds:si-es:di)+1, if length > 64K then err

PUSH    ES
PUSH    DI                  ;Save Start Address ES:DI
PUSH    CX                  ;Save length

CALL    GET5DIGITS          ;For Destination, get (up to) 20 bit parameter. 16 bit value (4 digits) to DI.
MOV     SI,DI                ;Put destination address in DS:SI
MOV     AX,ES
MOV     DS,AX                ;If 5 digits, then the first digit is put in DS

POP     CX                  ;Get length
POP     DI                  ;Get start ES:DI  destination DS:SI
POP     ES                  ;Get back the initial ES value (often 0)

MOVE1:  MOV     AL,[ES:DI]    ;Note cannot use MOVS opcode because of segment boundaries
MOV     [DS:SI],AL
CALL    Inc_DI_boundry_check ;Check if we are crossing a segment boundry

```

```

        CALL    Inc_SI_boundry_check
MOVE3: LOOP    MOVE1
        RET

```

```

;-----SUBSTITUTE Memory -----

```

```

SUBSTITUTE:

```

```

        CALL    GET5DIGITS          ;Get (up to) 20 bit parameter. 16 bit value (4 digits) to DI.
                                         ;If 5 digits, then the first digit is put in ES (highest nibble)

```

```

nusloop:CALL    CRLF

```

```

        call    SHOW_ADDRESS_ES
        mov     cx,8                 ;Display 8 bytes per line

```

```

sloop: call    BLANK

```

```

        mov     al,[es:DI]

```

```

        push    cx

```

```

        push    ax

```

```

        call    AL_HEXOUT

```

```

        mov     cl,'-'

```

```

        call    CO

```

```

        pop     ax

```

```

        pop     cx

```

```

        call    GET2DIGITS          ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged), terminator in AH

```

```

        cmp     ah,CR               ;CR signals we are done

```

```

je      qtest
cmp     ah,ESC           ;Also ESC
je      qtest
cmp     ah,' '           ;is a SP so skip to next byte
je      snext1
mov     [es:DI],al
snext1:  inc     DI
        cmp     DI,0
        jnz     snext2
        mov     AX,ES
        add     AX,1000H
        mov     ES,AX
snext2:  loop    sloop
        jmp     nusloop
qtest:  ret

```

;-----Verify Memory Contents -----

```

VERIFY:  CALL    GET5DIGITS           ;Get (up to) 20 bit parameter. 16 bit value (4 digits) to DI.
                                           ;If 5 digits, then the first digit is put in ES (highest nibble)
        push    ES                   ;Do everything relative to first ES value
        push    DI                   ;Save start address for now.

```

```

CALL    GET5DIGITS

MOV     SI,DI                ;Put end address in DS:DI

MOV     AX,ES

MOV     DS,AX                ;If 5 digits, then the first digit is put in DS

POP     DI

POP     ES                    ;Start=ES:SI  End=DS:DI

call    CLENGTH                ;Length cx = (ds:si-es:di)+1, if length > 64K then err

PUSH    ES

PUSH    DI                    ;Save Start Address

PUSH    CX                    ;Save length

CALL    GET5DIGITS            ;For Destination, get (up to) 20 bit parameter. 16 bit value (4 digits) to DI.

MOV     SI,DI                ;Put destination address in DS:SI

MOV     AX,ES

MOV     DS,AX                ;If 5 digits, then the first digit is put in DS

POP     CX

POP     DI

POP     ES                    ;Get back the initial ES value (often 0)

MOV     BX,0                  ;Count of mis-matches

```

```

VERIFY1:MOV  AL,[ES:DI]          ;cannot use cmps because of segments
        CMP   AL,[DS:SI]
        JZ    MATCH_OK
        call  verr

MATCH_OK:
        INC   DI
        CMP   DI,0              ;Check if we are crossing a segment boundry
        JNZ   VERIFY2
        MOV   AX,ES
        ADD   AX,1000H
        MOV   ES,AX

VERIFY2:INC  SI
        CMP   SI,0              ;Check if we are crossing a segment boundry
        JNZ   VERIFY3
        MOV   AX,DS
        ADD   AX,1000H
        MOV   DS,AX

VERIFY3:LOOP VERIFY1
        CMP   BX,0              ;Was there any errors
        JNZ   TOTAL_MISMATCHES
        MOV   BX,MATCHES_OK
        CALL  PRINT_STRING

TOTAL_MISMATCHES:

```

RET

verr: CMP BX,0 ;Save count, print error

JNZ SKIP_DIFF_MSG

PUSH DS

MOV AX,CS

MOV DS,AX

MOV BX,DIFF_Header_Msg

CALL PRINT_STRING

POP DS

SKIP_DIFF_MSG:

CALL CRLF ;There is a mis-match show values

call SHOW_ADDRESS_ES

PUSH CX

MOV CX,6

call TABS

mov al,[ES:DI]

PUSH AX

call AL_HEXOUT

CALL BLANK

POP AX

call AL_BINOUT

```

MOV     CX,5
call    TABS
call    SHOW_ADDRESS_DS

```

```

MOV     CX,6
call    TABS
mov     al,[ds:SI]
PUSH    AX
call    AL_HEXOUT
CALL    BLANK
POP     AX
call    AL_BINOUT

```

```

POP     CX
call    CTRL_CHECK
INC     BX                ;This prevents the header being show each time
RET                                ;Return verr

```

;----- Simple test of RAM (Continous)-----

TEST_RAM:

```

mov     bx,JMSG                ;Will test memory forever

```



```

call    PRINT_STRING

CALL    GET5DIGITS          ;Get (up to) 20 bit parameter. 16 bit value (4 digits) to DI.
                                ;If 5 digits, first digit entered to ES (Highest nibble)

PUSH    ES
PUSH    DI                  ;Save start address for now.

CALL    GET5DIGITS
MOV     SI,DI               ;Put end address in DS:SI
MOV     AX,ES
MOV     DS,AX              ;If 5 digits, then the first digit is put in DS

POP     DI
POP     ES                  ;Start=ES:BX  End=DS:DX

CALL    CLENGTH             ;Length cx = (ds:SI-es:DI)+1, if length > 64K then err

MOV     DX,0               ;Test loop count

PUSH    CX                  ;CX has length
mov     bx,STARTJMSG        ;Test memory until ESC
call    PRINT_STRING
POP     CX

```

```

mtest1:    push    cx

```

```

        push    di

mtloop:    mov     al,[es:DI]

        mov     ah,al                ;Store value currently in RAM
        not     al
        mov     [es:DI],al
        mov     al,[es:DI]
        not     al
        cmp     al,ah
        jne     terr
        mov     [es:DI],ah

tnext: inc    DI
        CMP     DI,0
        JNZ     tnext2
        mov     AX,ES
        ADD     ax,1000H
        MOV     ES,AX

tnext2:    call    CTRL_CHECK        ;See if an abort is requested
        loop    mtloop              ;Repeat for "length" number of bytes

        mov     bx,RAM_Test_Count
        CALL    PRINT_STRING
        inc     dx
        MOV     AX,DX
        CALL    AX_HEXOUT

```

```

mov     bx,H_MSG           ;H.
CALL    PRINT_STRING
pop     di                 ;Repeat the whole process
pop     cx
jmp     mtest1             ;test forever

```

```

terr:   push    DX
        mov     dx,ax       ;save data in dx
        CALL    CRLF
        call    SHOW_ADDRESS_ES
        mov     ax,dx       ;get back data
        xor     al,ah       ;identify bits
        mov     dx,ax
        call    AL_HEXOUT
        call    BLANK
        mov     ax,dx
        call    AL_BINOUT
        pop     dx
        jmp     tnext

```

```

;----- QUERY PORTS -----

```

```

QUERY: call  CICO                ;is it input or output

        cmp   al,'I'
        jz    input
        cmp   al,'O'
        jz    output
        jmp   ERR                ;if not QI or QO then error


input: call  GET4DIGITS           ;Get 8 or 16 bit value (2 or 4 digits) to DI, terminator in AH
        call  CRLF
        mov   dx,DI
        in    al,dx              ;Note will assume here we have just an 8 bit port
        push  ax
        call  AL_HEXOUT          ;Show value in HEX
        call  BLANK
        pop   ax
        call  AL_BINOUT          ;Show value in binary
        ret


output:   call  GET4DIGITS           ;Get 8 or 16 bit value (2 or 4 digits) to DI, terminator in AH
        mov   dx,DI
        CALL  GET2DIGITS          ;Output value to AL (BX unaltered)
        PUSH  AX
        CALL  CRLF

```

```

POP    AX

out    dx,al                ;Send 8 bit value in AL to port at [DX]

RET

```

```

;----- GO TO A RAM LOCATION -----

```

```

GOTO:  mov    bx,GET_SEG_MSG        ;Segment=

        call  PRINT_STRING

        call  GET4DIGITS            ;Get (up to) 16 bit value (4 digits) to BX.

        PUSH  DI                    ;Save Segment (in [DI]) on stack)


        mov    bx,GET_OFFS_MSG      ;Offset=

        call  PRINT_STRING

        call  GET4DIGITS

        PUSH  DI                    ;Save Offset (in [DI]) on stack)

        RETF                        ;Will pop offset, then CS and go there Note RETF!

```

```

;----- SWITCH CONTROL BACK TO Z80 (Master) -----

```

```

Z80:  in    al,SW86                ;This switches control back over to Z80
      MOV    AL,00000000B         ;Or reset TMA-O back to Z80 control.
      OUT    SW86_TM,AL
      nop
      nop
      nop
      nop
      nop
      JMP    BEGIN

```

```

;----- HEX MATH -----

```

```

HEXMATH:
      mov    bx,MATH_MSG          ;HEX MATH
      call   PRINT_STRING
      call   GET4DIGITS
      push   DI                   ;save data for the moment
      call   GET4DIGITS
      push   DI
      mov    bx,MATH_HEADER1
      call   PRINT_STRING
      pop    DI                  ;get back data2 (DI=data2)

```

```

pop     BX                ;and data1  (BX=data1)

push    DI                ;save them again for below
push    BX
add     BX,DI
call    BX_HEXOUT         ;Show addition (data1+data2)
mov     bx,MATH_HEADER2
call    PRINT_STRING
pop     BX                ;get back data1 one more time
pop     DI                ;and data2
sub     bx,DI             ;data1-data2
call    BX_HEXOUT
MOV     BX,H_MSG_CRLF
call    PRINT_STRING
ret

```

```

;----- Display all active IO input ports in the system -----
;
;          64K of 16 & 8 bit ports

```

```

INPORTS:mov  bx,PORTS_IN_MSG
           call PRINT_STRING
MOV     SI,0FFFFH        ;Display 12 lines, 4 ports across
MOV     BP,0000H         ;Will contain port number

```

```

LOOPPIO:      MOV     DX,BP

              CMP     DL,SW86                ;INPUT FROM THIS PORT SWITCHES THE 80286 BACK TO THE Z80 (SKIP)
              JZ      NEXT_P

              CMP     DL,SW68K              ;INPUT FROM THIS PORT SWITCHES IN THE 68K CPU (Do not activate)
              JZ      NEXT_P

              IN       AX,DX                ;Is it 0FFFFH
              CMP     AX,0FFFFH
              JZ      NEXT_P                ;If so skip

              MOV     AX,BP
              CALL    AX_HEXOUT             ;Print Port Address
              MOV     CL,'-'                ;Put in a "->"
              CALL    CO
              MOV     CL,'>'
              CALL    CO

              MOV     DX,BP
              IN       AX,DX                ;Get port WORD data (note, 8 bits even port Input will be in AH)
              CALL    AX_HEXOUT
              CALL    BLANK
              MOV     CL,'-'                ;Put in a "->"
              CALL    CO
              MOV     CL,'>'
              CALL    CO

```



```

MOV     DX,BP

IN      AL,DX                ;Get port BYTE data

CALL    AL_HEXOUT

SKIP2:  CALL    BLANK        ;4 ports across

CALL    BLANK

INC     SI                    ;Next cursor position count

MOV     AX,SI

AND     AX,0003H

CMP     AX,0003H

JNZ     NEXT_P

CALL    CRLF

NEXT_P:  INC     BP            ;Next Port until 0000H

OR      BP,BP

JZ      SKIP3

MOV     AX,BP

AND     AX,00FFH            ;256 ports at a time

JNZ     LOOPIO

mov     BX,MORE_MSG         ;Contineue (Y/N)

call    PRINT_STRING

call    CICO

cmp     al,'Y'

JNZ     SKIP3

```

```

CALL    CRLF
JMP     LOOPIO
SKIP3:  CALL    CRLF          ;Routine done/aborted
RET

```

```

;-----

```

```

REGISTERS:          ;RM Display all the CPU Registers

PUSH    AX          ;Save everything
PUSH    BX
PUSH    CX
PUSH    DX
PUSH    SI
PUSH    DI
PUSH    BP

PUSHF
PUSHF

PUSH    SP          ;we will display reverse this order
PUSH    BP
PUSH    DI
PUSH    SI

PUSH    DX

```

PUSH CX

PUSH BX

PUSH AX

MOV BX, INT_AX_MSG ; "AX="

CALL PRINT_STRING

POP AX

CALL AX_HEXOUT

MOV BX, INT_BX_MSG ; "H BX="

CALL PRINT_STRING

POP AX

CALL AX_HEXOUT

MOV BX, INT_CX_MSG ; "H CX="

CALL PRINT_STRING

POP AX

CALL AX_HEXOUT

MOV BX, INT_DX_MSG ; "H DX="

CALL PRINT_STRING

POP AX

CALL AX_HEXOUT

```

MOV     BX,INT_SI_MSG           ;"SI="
CALL    PRINT_STRING
POP     AX
CALL    AX_HEXOUT

MOV     BX,INT_DI_MSG           ;"DI="
CALL    PRINT_STRING
POP     AX
CALL    AX_HEXOUT

MOV     BX,INT_BP_MSG           ;"BP="
CALL    PRINT_STRING
POP     AX
CALL    AX_HEXOUT

MOV     BX,INT_SP_MSG           ;"SP="
CALL    PRINT_STRING
POP     AX
ADD     AX,14                   ; Adjust because we first saved stuff above (PUSHAD)
CALL    AX_HEXOUT

MOV     BX,INT_FLAGS_MSG        ;[Flags]=
CALL    PRINT_STRING
POP     AX                     ;Flags to AX
MOV     AL,AH

```

```
CALL    AL_BINOUT          ;Output lower 16 bits
POP     AX
CALL    AL_BINOUT

MOV     BX,INT_CS_MSG      ;"CS="
CALL    PRINT_STRING
MOV     AX,CS
CALL    AX_HEXOUT

MOV     BX,INT_DS_MSG      ;"DS="
CALL    PRINT_STRING
MOV     AX,DS
CALL    AX_HEXOUT

MOV     BX,INT_ES_MSG      ;"ES="
CALL    PRINT_STRING
MOV     AX,ES
CALL    AX_HEXOUT

MOV     BX,INT_SS_MSG      ;"SS="
CALL    PRINT_STRING
MOV     AX,SS
CALL    AX_HEXOUT

MOV     BX,H_MSG_CRLF
```

```
CALL    PRINT_STRING
```

```
POP     BP                ;Get Back Everything
```

```
POP     DI
```

```
POP     SI
```

```
POP     DX
```

```
POP     CX
```

```
POP     BX
```

```
POP     AX
```

```
RET
```

```
;----- THIS IS A ROUTINE TO SET THE CURRENT TIME & DATE FOR THE DALLAS RTC CHIP ON THE MS-DOS SUPPORT BOARD -----
```

```
SET_TIME_DATE:
```

```
CALL    RAW_GETTIME      ;First Show Time/Date
```

```
CALL    CRLF             ;First set Time
```

```
MOV     BX,SET_TIME_MSG
```

```
CALL    PRINT_STRING
```

```
CALL    CICO
```

```
CMP     AL,'Y'
```

```
JZ      SET_TIME0
```

```
RET
```

SET_TIME0:

```

CALL    InputTime          ;CH = HOURS, CL = Minutes, DH = Seconds all in BCD
CALL    UPD_IN_PR          ;CHECK FOR UPDATE IN PROCESS
JNC     SET_TIME1          ;GO AROUND IF OK
JMP     RTC_ERROR          ;IF ERROR

```

SET_TIME1:

```

PUSH    DX                 ;Save Data
MOV     DL,-2              ;-2 goes to 0 for PORT_INC_2
CLI                                           ;INTERRUPTS OFF DURING WRITES

```

```

CALL    PORT_INC_2         ;SET ADDRESS OF SECONDS

```

```

POP     DX

```

```

MOV     AL,DH

```

```

OUT     CMOS_PORT+1,AL     ;Send Seconds

```

```

CALL    PORT_INC_2         ;SET ADDRESS OF MINUTES

```

```

MOV     AL,CL

```

```

OUT     CMOS_PORT+1,AL     ;Send Minutes

```

```

CALL    PORT_INC_2         ;SET ADDRESS OF HOURS

```

```

MOV     AL,CH

```

```

OUT     CMOS_PORT+1,AL     ;Send Hours

```

```

CALL    CRLF

                                ;Now Set date

CALL    UPD_IN_PR              ;CHECK FOR UPDATE IN PROCESS

JNC     SET_DATE1              ;GO AROUND IF OK

JMP     RTC_ERROR              ;IF ERROR

```

SET_DATE1:

```

CALL    InputDate              ;CH = CENTURY, CL = Year, DH = Month, DL = Day all in BCD

PUSH    DX                    ;Save 2X

PUSH    DX

MOV     DL,6

CLI                                           ;INTERRUPTS OFF DURING WRITES


CALL    PORT_INC               ;SET ADDRESS OF DAYS (Port 7)

POP     DX

MOV     AL,DL

OUT     CMOS_PORT+1,AL          ;Send Days


CALL    PORT_INC               ;SET ADDRESS OF MONTHS

POP     DX

MOV     AL,DH

OUT     CMOS_PORT+1,AL          ;Send Months


CALL    PORT_INC               ;SET ADDRESS OF YEARS

MOV     AL,CL

```



```

OUT    CMOS_PORT+1,AL          ;Send Seconds

MOV    DL,31H                  ;POINT TO CENTURY BYTE SAVE AREA
CALL   PORT_INC                ;
MOV    AL,CH
OUT    CMOS_PORT+1,AL          ;Send Century

JMP    RAW_GETTIME

```

;----- THIS IS A ROUTINE TO PLACE TIME & DATE ON CONSOLE AT THE CURRENT CURSOR POSITION -----

RAW_GETTIME:

```

CALL   CRLF
MOV    BX,Time_Msg
CALL   PRINT_STRING
CALL   LOAD_TIME
CALL   DisplayTime

MOV    BX,GAP_Msg
CALL   PRINT_STRING

MOV    BX,Date_Msg

```

```

CALL PRINT_STRING
CALL LOAD_DATE
CALL DisplayDate
RET

```

```

LOAD_TIME:                ;Load up registers with TIME info

CALL UPD_IN_PR            ;CHECK FOR UPDATE IN PROCESS

JNC RTC_2A                ;GO AROUND IF OK

JMP RTC_ERROR             ;IF ERROR


RTC_2A:    MOV     DL,-2                ;-2 goes to 0 for PORT_INC_2

CLI                ;INTERRUPTS OFF DURING READ

CALL PORT_INC_2      ;SET ADDRESS OF SECONDS

IN     AL,CMOS_PORT+1        ;Get BCD value returned

MOV    DH, AL              ;SAVE IN DH

CALL PORT_INC_2      ;SET ADDRESS OF MINUTES

IN     AL,CMOS_PORT+1        ;Get BCD value returned

MOV    CL,AL              ;SAVE IN CL

CALL PORT_INC_2      ;SET ADDRESS OF HOURS

IN     AL,CMOS_PORT+1        ;Get BCD value returned

STI

MOV    CH,AL              ;SAVE

MOV    DL,0              ;SET DL TO ZERO

CLC                ;Clear carry flag to indicate all is OK

RET

```

```

LOAD_DATE:                ;Load up registers with Date info

    CALL  UPD_IN_PR        ;CHECK FOR UPDATE IN PROCESS

    JNC   RTC_2B           ;GO AROUND IF OK

    JMP   RTC_ERROR        ;IF ERROR


RTC_2B:    MOV     DL,6

    CLI                     ;INTERRUPTS OFF DURING READ

    CALL  PORT_INC         ;POINT TO DAY

    IN    AL,CMOS_PORT+1

    MOV   CH,AL            ;TEMPORY SAVE HERE (Return in DL)


    CALL  PORT_INC         ;POINT TO MONTH

    IN    AL,CMOS_PORT+1

    MOV   DH,AL            ;SAVE


    CALL  PORT_INC         ;POINT TO YEAR

    IN    AL,CMOS_PORT+1

    MOV   CL,AL            ;SAVE

    MOV   DL,31H           ;POINT TO CENTURY BYTE SAVE AREA

    CALL  PORT_INC         ;

    IN    AL,CMOS_PORT+1    ;GET VALUE

    STI

    MOV   DL,CH            ;GET DAY BACK

    MOV   CH,AL

```

```
CLC                ;Clear carry flag to indicate all is OK
```

```
RET
```

```
RTC_ERROR:
```

```
MOV    BX,TIME_ERROR_MSG
```

```
CALL   PRINT_STRING
```

```
STI
```

```
STC                ;Set carry flag to indicate all is NOT OK
```

```
RET                ;Back to main menu
```

```
PORT_INC:
```

```
INC    DL          ;INCREMENT ADDRESS
```

```
MOV    AL,DL
```

```
OUT    CMOS_PORT,AL
```

```
RET
```

```
PORT_INC_2:
```

```
ADD    DL,2        ;INCREMENT ADDRESS
```

```
MOV    AL,DL
```

```
OUT    CMOS_PORT,AL
```

```
RET
```

```
INITIALIZE_STATUS: ;Initilize the RTC
```

```
PUSH   DX          ;SAVE
```

```
MOV    DL,09H
```

```

CLI                                ;INTERRUPTS MASKED DURING RESET

CALL  PORT_INC

MOV   AL,26H

OUT   CMOS_PORT+1,AL              ;INITIALIZE 'A' REGISTER

CALL  PORT_INC

MOV   AL,82H                      ;SET 'SET BIT' FOR CLOCK INITIALIZATION
                                         ;AND 24 HOUR MODE

OUT   CMOS_PORT+1,AL              ;INITIALIZE 'B' REGISTER

CALL  PORT_INC

IN    AL,CMOS_PORT+1              ;READ REGISTER 'C' TO INITIALIZE

CALL  PORT_INC

IN    AL,CMOS_PORT+1              ;READ REGISTER 'D' TO INITIALIZE

STI

POP   DX                          ;RESTORE

RET

```

```

UPD_IN_PR:                          ;Check we are ready to read clock

    PUSH  CX

    MOV   CX,600                   ;SET LOOP COUNT

UPDATE:    MOV   AL,0AH              ;ADDRESS OF [A] REGISTER

    CLI                                ;INTERRUPTS MASKED DURING RESET

    OUT   CMOS_PORT,AL

    JMP   $+2                      ;I/O TIME DELAY

    IN    AL,CMOS_PORT+1            ;READ IN REGISTER [A]

```

```

    TEST    AL,80H                ;IF 8XH--> UIP BIT  IS ON (CANNOT READ TIME)

    JZ      UPD_IN_PREND

    LOOP    UPDATE                ;Try again

    STC                      ;SET CARRY FOR ERROR

    XOR     AX,AX                ;

UPD_IN_PREND:

    POP     CX

    RET                      ;RETURN

```

```

;Display time

```

```

;    Arrive with CH = HOURS IN BCD

;                CL = Minutes in BCD

;                DH = Seconds in BCD

```

```

DisplayTime:

```

```

    PUSH    BX

    PUSH    DX

    PUSH    CX

    MOV     AL,CH

    CALL    PRINT_REG            ;Hours.  Convert BCD to ASCII

    MOV     CL,':'

    CALL    CO

    POP     CX

    MOV     AL,CL

    CALL    PRINT_REG            ;Minutes.  Convert BCD to ASCII

```

```

MOV     CL, ':'
CALL    CO
POP     DX
MOV     AL, DH
CALL    PRINT_REG           ;Seconds.  Convert BCD to ASCII
POP     BX
RET

```

;Input time

```

;      Return CH = HOURS IN BCD
;
;      CL = Minutes in BCD
;
;      DH = Seconds in BCD

```

InputTime:

```

PUSH    BX
MOV     BX, Input_Hours_Msg
CALL    PRINT_STRING
CALL    GET2BCD             ;Return with 2 BCD digits in AL
MOV     CH, AL
PUSH    CX
MOV     BX, Input_Minutes_Msg
CALL    PRINT_STRING
CALL    GET2BCD             ;Return with 2 BCD digits in AL
POP     CX
MOV     CL, AL
PUSH    CX

```

```

MOV     BX,Input_Seconds_Msg
CALL    PRINT_STRING
CALL    GET2BCD                ;Return with 2 BCD digits in AL
MOV     DH,AL
PUSH    DX
POP      DX
POP      CX
POP      BX
RET

```

;Display date

```

;      Return CH = CENTURY IN BCD
;
;      CL = Year in BCD
;
;      DH = Month in BCD
;
;      DL = Day in BCD

```

DisplayDate:

```

PUSH    BX
PUSH    DX
PUSH    DX
PUSH    CX
MOV     AL,CH
CALL    PRINT_REG              ;Century (19/20).  Convert BCD to ASCII
POP      CX
MOV     AL,CL

```



```

CALL PRINT_REG          ;Year.  Convert BCD to ASCII
MOV CL, '/'
CALL CO
POP DX
MOV AL, DH
CALL PRINT_REG          ;Month.  Convert BCD to ASCII
MOV CL, '/'
CALL CO
POP DX
MOV AL, DL
CALL PRINT_REG          ;Day.  Convert BCD to ASCII
POP BX
RET

```

```

PRINT_REG:              ;Print BCD in [AL]
    PUSH AX
    MOV CL, 4
    RCR AX, CL
    AND AL, 0FH
    ADD AL, 30H
    MOV CL, AL           ;Write high byte mins to CRT
    CALL CO
    POP AX
    AND AL, 0FH
    ADD AL, 30H

```

```
MOV    CL,AL
```

```
CALL   CO
```

```
RET
```

```
;Input Date
```

```
;      Return CH = CENTURY IN BCD
```

```
;          CL = Year in BCD
```

```
;          DH = Month in BCD
```

```
;          DL = Day in BCD
```

```
InputDate:
```

```
PUSH   BX
```

```
MOV     BX,Input_Year_Msg
```

```
CALL    PRINT_STRING
```

```
CALL    GET2BCD                ;Return with 2 BCD digits in AL
```

```
MOV     CL,AL
```

```
MOV     CH,20H                ;Assume 20 for century
```

```
PUSH    CX                    ;Save
```

```
MOV     BX,Input_Month_Msg
```

```
CALL    PRINT_STRING
```

```
CALL    GET2BCD                ;Return with 2 BCD digits in AL
```

```
MOV     DH,AL
```

```
PUSH    DX                    ;Save
```

```
MOV     BX,Input_Day_Msg
```

```
CALL    PRINT_STRING
```

```

CALL    GET2BCD                ;Return with 2 BCD digits in AL
POP     DX                    ;Get back
MOV     DL,AL
POP     CX                    ;Get back
POP     BX
RET

```

```

GET2BCD:                        ;Return with 2 BCD digits in AL

```

```

CALL    CICO
mov     ah,0
CMP     AL,ESC                ;Abort if ESC
JNZ     BCD_OK
JMP     INIT                  ;Back to start of Monitor

```

```

BCD_OK:    SUB     AL,'@'

```

```

SHL     AL,1
SHL     AL,1
SHL     AL,1
SHL     AL,1
PUSH    AX
CALL    CICO
SUB     AL,'@'
AND     AL,0FH
MOV     CL,AL

```

```

POP     AX
OR      AL,CL
RET

```

```

;----- Run diagnostic tests on the 8259A PIC. -----
;Configured below for the S100Computers PIC/RTC S-100 and MSDOS Support Boards
;We will fill out all 256 Interrupt vectors with a diagnostic routine to show
;what interrupt was triggered if not the 8259A bit 1 int.

```

```

TEST_8259:                                ;"L" Main menu option
      mov     bx,PIC_SIGNON                ;Send a 8259A Test signon message
      call    PRINT_STRING

      CALL    SETUP_INT_TABLE              ;Setup Int table (0-400H in RAM)

      mov     ax,cs                        ;Note this is just a simplified sub-section of the SETUP_IBM_BIOS routine
      mov     ds,ax                        ;DS is this ROM's CS
      sub     ax,ax
      mov     es,ax                        ;ES: = 0H in RAM for STOW's below, DS: = CS:(here).
      CLD                                  ;Default to direction up

      mov     di,3fcH                      ;Int FFH seems to false trigger on 80386 board (not 8086 board!)
      mov     ax,dummy_return              ;Have it point to Dummy return in this monitor

```

```

        stosw                ;(ES: used for final location)

mov     cx,8                ;Set all 8 hardware interrupts for 8259A (at I/O port address 20H)
mov     si,vec_tbl_8259A    ;Move the pointers in vec_tbl-8259A to low RAM starting at 20H
mov     di,Start8259A_Ints  ;Note DS:(=CS:) is source,   ES: is destination
T2_8259:movsw
        inc     di          ;Skip over the segment pointer (already done above), to next vector offset
        inc     di
        loop    T2_8259

mov     cx,16               ;Set all 16 MS-DOS software interrupts locations
mov     si,vec_tbl_soft_ints
mov     di,CRTINT          ;Start location in low RAM
T3_8259:movsw               ;Note DS: (=CS:) is source,   ES: is destination
        inc     di          ;Skip over the segment pointer (already done above), to next vector offset
        inc     di          ;Note this sets up the VID_PARAMATER_TABLES & INITIAL FLOPPY PARMS TABLE as well
        loop    T3_8259

mov     ax,keybuff         ;Keyboard buffer interrupt pointers
mov     [es:bufhd],ax       ;Head of buffer = end of buffer
mov     [es:buftl],ax       ;buffer end
mov     byte [es:chrcnt],0   ;Character count

mov     al,11111111b        ;Block all INTs initially
out     MASTER_PIC_PORT+1,al

```

```

mov    al,MasterICW1           ;Initilize the 8259A PIC Controller
out    MASTER_PIC_PORT,al
mov    al,MasterICW2           ;Ints starts at 20H in RAM
out    MASTER_PIC_PORT+1,al
mov    al,MasterICW4           ;No slaves above, so 8259 does not expect ICW3
out    MASTER_PIC_PORT+1,al

mov    al,11111101b           ;Allow V1 (ONLY) on 8259A
out    MASTER_PIC_PORT+1,al
sti                               ;Enable hardware interrupts

```

T51_8259:

```

MOV    BX,IN_CHAR_MSG          ;'Type one character'
CALL   PRINT_STRING

```

T5_8259:

```

MOV    AH,01H                  ;Check if anything there
int     16H                     ;Get Keyboard status. Console Input Handler    (Software Interrupt 16H)
JZ     T5_8259

MOV    AH,0H                   ;Get actual character from buffer
int     16H                     ;Get Character. Console Input Handler    (Software Interrupt 16H)

CMP    AL,ESC

```

```

JZ      T6_8259                      ;We are done if an ESC character
MOV     CL,AL
CALL    CO                          ;Display character recieved
sti                                           ;Ints back on
JMP     T5_8259

```

T6_8259:

```

mov     al,11111111b                 ;Do not Allow V1 on 8259A again
out     MASTER_PIC_PORT+1,al
cli                                           ;Turn hardware int's back off
JMP     ToMonitor

```

;----- INTERRUPT MODE JUMP TABLE -----

; Note some of the code below is for the 80286 (and 80386 CPU's).

```

RM_INT_JUMP_TABLE:                    ;Unfortunately we have to do all 256 possible routines!
DB      6AH,0H                        ;Quirk with NASM forces 80H and above to a word so use DB's
jmp     word RM_Zero_INT_Routine ;0  Divide by 0
DB      6AH,1H                        ; In every case below we push a byte on the stack to identify the INT
jmp     word RM_TRACE_INT_Routine ;1  CPU Trace Interrupt
DB      6AH,2H
jmp     word RM_NMI_INT_Routine  ;2  NMI default INT
DB      6AH,3H

```

```

jmp    word RM_CC_INT_Routine          ;3  Software CC Interrupt
DB     6AH,4H

jmp    word RM_Overflow_INT_Routine    ;4  Overflow INT
DB     6AH,5H

jmp    word RM_Bounds_INT_Routine      ;5  Bounds Check, FAULT
DB     6AH,6H

jmp    word RM_Opcode_INT_Routine      ;6  Invalid Opcode, FAULT
DB     6AH,7H

jmp    word RM_Device_INT_Routine      ;7  Device not available, FAULT
DB     6AH,8H

jmp    word RM_DFault_INT_Routine      ;8  Double fault Fault
DB     6AH,9H

jmp    word RM_MathSeg_INT_Routine      ;9  Math Coprocessor Segment error
DB     6AH,0AH

jmp    word RM_TSS_INT_Routine         ;10 Invalid TSS (+ Error Number)
DB     6AH,0BH

jmp    word RM_Segment_INT_Routine     ;11 Segment Error (+ Error Number)
DB     6AH,0CH

jmp    word RM_Stack_INT_Routine       ;12 Stack Exception (+ Error Number)
DB     6AH,0DH

jmp    word RM_General_INT_Routine     ;13 General Protection (+ Error Number)
DB     6AH,0EH

jmp    word RM_Page_INT_Routine        ;14 Page error, FAULT
DB     6AH,0FH

jmp    word RM_Intel_INT_Routine       ;15 Intel reserved TRAP

```



```
DB      6AH,10H

jmp     word RM_Coprocessor_INT_Routine ;16 Co-processor error, FAULT

DB      6AH,11H

jmp     word RM_Default_INT_Routine

DB      6AH,12H

jmp     word RM_Default_INT_Routine

DB      6AH,13H

jmp     word RM_Default_INT_Routine

DB      6AH,14H

jmp     word RM_Default_INT_Routine

DB      6AH,15H

jmp     word RM_Default_INT_Routine

DB      6AH,16H

jmp     word RM_Default_INT_Routine

DB      6AH,17H

jmp     word RM_Default_INT_Routine

DB      6AH,18H

jmp     word RM_Default_INT_Routine

DB      6AH,19H

jmp     word RM_Default_INT_Routine

DB      6AH,1AH

jmp     word RM_Default_INT_Routine

DB      6AH,1BH

jmp     word RM_Default_INT_Routine
```

```
DB      6AH,1CH
jmp     word RM_Default_INT_Routine
DB      6AH,1DH
jmp     word RM_Default_INT_Routine
DB      6AH,1EH
jmp     word RM_Default_INT_Routine
DB      6AH,1FH
jmp     word RM_Default_INT_Routine

DB      6AH,20H
jmp     word RM_Default_INT_Routine
DB      6AH,21H
jmp     word RM_Default_INT_Routine
DB      6AH,22H
jmp     word RM_Default_INT_Routine
DB      6AH,23H
jmp     word RM_Default_INT_Routine
DB      6AH,24H
jmp     word RM_Default_INT_Routine
DB      6AH,25H
jmp     word RM_Default_INT_Routine
DB      6AH,26H
jmp     word RM_Default_INT_Routine
DB      6AH,27H
jmp     word RM_Default_INT_Routine
```

```
DB      6AH,28H
jmp     word RM_Default_INT_Routine
DB      6AH,29H
jmp     word RM_Default_INT_Routine
DB      6AH,2AH
jmp     word RM_Default_INT_Routine
DB      6AH,2BH
jmp     word RM_Default_INT_Routine
DB      6AH,2CH
jmp     word RM_Default_INT_Routine
DB      6AH,2DH
jmp     word RM_Default_INT_Routine
DB      6AH,2EH
jmp     word RM_Default_INT_Routine
DB      6AH,2FH
jmp     word RM_Default_INT_Routine

DB      6AH,30H
jmp     word RM_Default_INT_Routine
DB      6AH,31H
jmp     word RM_Default_INT_Routine
DB      6AH,32H
jmp     word RM_Default_INT_Routine
DB      6AH,33H
jmp     word RM_Default_INT_Routine
```

```
DB      6AH,34H
jmp     word RM_Default_INT_Routine
DB      6AH,35H
jmp     word RM_Default_INT_Routine
DB      6AH,36H
jmp     word RM_Default_INT_Routine
DB      6AH,37H
jmp     word RM_Default_INT_Routine
DB      6AH,38H
jmp     word RM_Default_INT_Routine
DB      6AH,39H
jmp     word RM_Default_INT_Routine
DB      6AH,3AH
jmp     word RM_Default_INT_Routine
DB      6AH,3BH
jmp     word RM_Default_INT_Routine
DB      6AH,3CH
jmp     word RM_Default_INT_Routine
DB      6AH,3DH
jmp     word RM_Default_INT_Routine
DB      6AH,3EH
jmp     word RM_Default_INT_Routine
DB      6AH,3FH
jmp     word RM_Default_INT_Routine
```

```
DB      6AH,40H
jmp     word RM_Default_INT_Routine
DB      6AH,41H
jmp     word RM_Default_INT_Routine
DB      6AH,42H
jmp     word RM_Default_INT_Routine
DB      6AH,43H
jmp     word RM_Default_INT_Routine
DB      6AH,44H
jmp     word RM_Default_INT_Routine
DB      6AH,45H
jmp     word RM_Default_INT_Routine
DB      6AH,46H
jmp     word RM_Default_INT_Routine
DB      6AH,47H
jmp     word RM_Default_INT_Routine
DB      6AH,48H
jmp     word RM_Default_INT_Routine
DB      6AH,49H
jmp     word RM_Default_INT_Routine
DB      6AH,4AH
jmp     word RM_Default_INT_Routine
DB      6AH,4BH
jmp     word RM_Default_INT_Routine
DB      6AH,4CH
```

```
jmp    word RM_Default_INT_Routine
DB     6AH,4DH

jmp    word RM_Default_INT_Routine
DB     6AH,4EH

jmp    word RM_Default_INT_Routine
DB     6AH,4FH

jmp    word RM_Default_INT_Routine

DB     6AH,50H

jmp    word RM_Default_INT_Routine
DB     6AH,51H

jmp    word RM_Default_INT_Routine
DB     6AH,52H

jmp    word RM_Default_INT_Routine
DB     6AH,53H

jmp    word RM_Default_INT_Routine
DB     6AH,54H

jmp    word RM_Default_INT_Routine
DB     6AH,55H

jmp    word RM_Default_INT_Routine
DB     6AH,56H

jmp    word RM_Default_INT_Routine
DB     6AH,57H

jmp    word RM_Default_INT_Routine
DB     6AH,58H
```

```
jmp    word RM_Default_INT_Routine
DB     6AH,59H

jmp    word RM_Default_INT_Routine
DB     6AH,5AH

jmp    word RM_Default_INT_Routine
DB     6AH,5BH

jmp    word RM_Default_INT_Routine
DB     6AH,5CH

jmp    word RM_Default_INT_Routine
DB     6AH,5DH

jmp    word RM_Default_INT_Routine
DB     6AH,5EH

jmp    word RM_Default_INT_Routine
DB     6AH,5FH

jmp    word RM_Default_INT_Routine

DB     6AH,60H

jmp    word RM_Default_INT_Routine
DB     6AH,61H

jmp    word RM_Default_INT_Routine
DB     6AH,62H

jmp    word RM_Default_INT_Routine
DB     6AH,63H

jmp    word RM_Default_INT_Routine
DB     6AH,64H
```

```
jmp    word RM_Default_INT_Routine
DB     6AH,65H

jmp    word RM_Default_INT_Routine
DB     6AH,66H

jmp    word RM_Default_INT_Routine
DB     6AH,67H

jmp    word RM_Default_INT_Routine
DB     6AH,68H

jmp    word RM_Default_INT_Routine
DB     6AH,69H

jmp    word RM_Default_INT_Routine
DB     6AH,6AH

jmp    word RM_Default_INT_Routine
DB     6AH,6BH

jmp    word RM_Default_INT_Routine
DB     6AH,6CH

jmp    word RM_Default_INT_Routine
DB     6AH,6DH

jmp    word RM_Default_INT_Routine
DB     6AH,6EH

jmp    word RM_Default_INT_Routine
DB     6AH,6FH

jmp    word RM_Default_INT_Routine
DB     6AH,70H
```



```
jmp    word RM_Default_INT_Routine
DB     6AH,71H

jmp    word RM_Default_INT_Routine
DB     6AH,72H

jmp    word RM_Default_INT_Routine
DB     6AH,73H

jmp    word RM_Default_INT_Routine
DB     6AH,74H

jmp    word RM_Default_INT_Routine
DB     6AH,75H

jmp    word RM_Default_INT_Routine
DB     6AH,76H

jmp    word RM_Default_INT_Routine
DB     6AH,77H

jmp    word RM_Default_INT_Routine
DB     6AH,78H

jmp    word RM_Default_INT_Routine
DB     6AH,79H

jmp    word RM_Default_INT_Routine
DB     6AH,7AH

jmp    word RM_Default_INT_Routine
DB     6AH,7BH

jmp    word RM_Default_INT_Routine
DB     6AH,7CH

jmp    word RM_Default_INT_Routine
```

```

DB      6AH,7DH
jmp     word RM_Default_INT_Routine
DB      6AH,7EH
jmp     word RM_Default_INT_Routine
DB      6AH,7FH
jmp     word RM_Default_INT_Routine

DB      6AH,80H                      ;Quirk with NASM forces 80H and above to a word!
jmp     word RM_Default_INT_Routine  ;So use DB's
DB      6AH,81H
jmp     word RM_Default_INT_Routine
DB      6AH,82H
jmp     word RM_Default_INT_Routine
DB      6AH,83H
jmp     word RM_Default_INT_Routine
DB      6AH,84H
jmp     word RM_Default_INT_Routine
DB      6AH,85H
jmp     word RM_Default_INT_Routine
DB      6AH,86H
jmp     word RM_Default_INT_Routine
DB      6AH,87H
jmp     word RM_Default_INT_Routine
DB      6AH,88H
jmp     word RM_Default_INT_Routine

```

```
DB      6AH,89H
jmp     word RM_Default_INT_Routine
DB      6AH,8AH
jmp     word RM_Default_INT_Routine
DB      6AH,8BH
jmp     word RM_Default_INT_Routine
DB      6AH,8CH
jmp     word RM_Default_INT_Routine
DB      6AH,8DH
jmp     word RM_Default_INT_Routine
DB      6AH,8EH
jmp     word RM_Default_INT_Routine
DB      6AH,8FH
jmp     word RM_Default_INT_Routine

DB      6AH,90H
jmp     word RM_Default_INT_Routine
DB      6AH,91H
jmp     word RM_Default_INT_Routine
DB      6AH,92H
jmp     word RM_Default_INT_Routine
DB      6AH,93H
jmp     word RM_Default_INT_Routine
DB      6AH,94H
jmp     word RM_Default_INT_Routine
```

```
DB      6AH,95H
jmp     word RM_Default_INT_Routine
DB      6AH,96H
jmp     word RM_Default_INT_Routine
DB      6AH,97H
jmp     word RM_Default_INT_Routine
DB      6AH,98H
jmp     word RM_Default_INT_Routine
DB      6AH,99H
jmp     word RM_Default_INT_Routine
DB      6AH,9AH
jmp     word RM_Default_INT_Routine
DB      6AH,9BH
jmp     word RM_Default_INT_Routine
DB      6AH,9CH
jmp     word RM_Default_INT_Routine
DB      6AH,9DH
jmp     word RM_Default_INT_Routine
DB      6AH,9EH
jmp     word RM_Default_INT_Routine
DB      6AH,9FH
jmp     word RM_Default_INT_Routine

DB      6AH,0A0H
jmp     word RM_Default_INT_Routine
```

```
DB      6AH,0A1H
jmp     word RM_Default_INT_Routine
DB      6AH,0A2H
jmp     word RM_Default_INT_Routine
DB      6AH,0A3H
jmp     word RM_Default_INT_Routine
DB      6AH,0A4H
jmp     word RM_Default_INT_Routine
DB      6AH,0A5H
jmp     word RM_Default_INT_Routine
DB      6AH,0A6H
jmp     word RM_Default_INT_Routine
DB      6AH,0A7H
jmp     word RM_Default_INT_Routine
DB      6AH,0A8H
jmp     word RM_Default_INT_Routine
DB      6AH,0A9H
jmp     word RM_Default_INT_Routine
DB      6AH,0AAH
jmp     word RM_Default_INT_Routine
DB      6AH,0ABH
jmp     word RM_Default_INT_Routine
DB      6AH,0ACH
jmp     word RM_Default_INT_Routine
DB      6AH,0ADH
```

```
jmp    word RM_Default_INT_Routine
DB     6AH,0AEH

jmp    word RM_Default_INT_Routine
DB     6AH,0AFH

jmp    word RM_Default_INT_Routine

DB     6AH,0B0H

jmp    word RM_Default_INT_Routine
DB     6AH,0B1H

jmp    word RM_Default_INT_Routine
DB     6AH,0B2H

jmp    word RM_Default_INT_Routine
DB     6AH,0B3H

jmp    word RM_Default_INT_Routine
DB     6AH,0B4H

jmp    word RM_Default_INT_Routine
DB     6AH,0B5H

jmp    word RM_Default_INT_Routine
DB     6AH,0B6H

jmp    word RM_Default_INT_Routine
DB     6AH,0B7H

jmp    word RM_Default_INT_Routine
DB     6AH,0B8H

jmp    word RM_Default_INT_Routine
DB     6AH,0B9H
```

```
jmp    word RM_Default_INT_Routine
DB     6AH,0BAH

jmp    word RM_Default_INT_Routine
DB     6AH,0BBH

jmp    word RM_Default_INT_Routine
DB     6AH,0BCH

jmp    word RM_Default_INT_Routine
DB     6AH,0BDH

jmp    word RM_Default_INT_Routine
DB     6AH,0BEH

jmp    word RM_Default_INT_Routine
DB     6AH,0BFH

jmp    word RM_Default_INT_Routine

DB     6AH,0C0H

jmp    word RM_Default_INT_Routine
DB     6AH,0C1H

jmp    word RM_Default_INT_Routine
DB     6AH,0C2H

jmp    word RM_Default_INT_Routine
DB     6AH,0C3H

jmp    word RM_Default_INT_Routine
DB     6AH,0C4H

jmp    word RM_Default_INT_Routine
DB     6AH,0C5H
```

```
jmp    word RM_Default_INT_Routine
DB     6AH,0C6H

jmp    word RM_Default_INT_Routine
DB     6AH,0C7H

jmp    word RM_Default_INT_Routine
DB     6AH,0C8H

jmp    word RM_Default_INT_Routine
DB     6AH,0C9H

jmp    word RM_Default_INT_Routine
DB     6AH,0CAH

jmp    word RM_Default_INT_Routine
DB     6AH,0CBH

jmp    word RM_Default_INT_Routine
DB     6AH,0CCH

jmp    word RM_Default_INT_Routine
DB     6AH,0CDH

jmp    word RM_Default_INT_Routine
DB     6AH,0CEH

jmp    word RM_Default_INT_Routine
DB     6AH,0CFH

jmp    word RM_Default_INT_Routine

DB     6AH,0D0H

jmp    word RM_Default_INT_Routine
DB     6AH,0D1H
```



```
jmp    word RM_Default_INT_Routine
DB     6AH,0D2H

jmp    word RM_Default_INT_Routine
DB     6AH,0D3H

jmp    word RM_Default_INT_Routine
DB     6AH,0D4H

jmp    word RM_Default_INT_Routine
DB     6AH,0D5H

jmp    word RM_Default_INT_Routine
DB     6AH,0D6H

jmp    word RM_Default_INT_Routine
DB     6AH,0D7H

jmp    word RM_Default_INT_Routine
DB     6AH,0D8H

jmp    word RM_Default_INT_Routine
DB     6AH,0D9H

jmp    word RM_Default_INT_Routine
DB     6AH,0DAH

jmp    word RM_Default_INT_Routine
DB     6AH,0DBH

jmp    word RM_Default_INT_Routine
DB     6AH,0DCH

jmp    word RM_Default_INT_Routine
DB     6AH,0DDH

jmp    word RM_Default_INT_Routine
```

```
DB      6AH,0DEH
jmp     word RM_Default_INT_Routine
DB      6AH,0DFH
jmp     word RM_Default_INT_Routine

DB      6AH,0E0H
jmp     word RM_Default_INT_Routine
DB      6AH,0E1H
jmp     word RM_Default_INT_Routine
DB      6AH,0E2H
jmp     word RM_Default_INT_Routine
DB      6AH,0E3H
jmp     word RM_Default_INT_Routine
DB      6AH,0E4H
jmp     word RM_Default_INT_Routine
DB      6AH,0E5H
jmp     word RM_Default_INT_Routine
DB      6AH,0E6H
jmp     word RM_Default_INT_Routine
DB      6AH,0E7H
jmp     word RM_Default_INT_Routine
DB      6AH,0E8H
jmp     word RM_Default_INT_Routine
DB      6AH,0E9H
jmp     word RM_Default_INT_Routine
```

```
DB      6AH,0EAH
jmp     word RM_Default_INT_Routine
DB      6AH,0EBH
jmp     word RM_Default_INT_Routine
DB      6AH,0ECH
jmp     word RM_Default_INT_Routine
DB      6AH,0EDH
jmp     word RM_Default_INT_Routine
DB      6AH,0EEH
jmp     word RM_Default_INT_Routine
DB      6AH,0EFH
jmp     word RM_Default_INT_Routine

DB      6AH,0F0H
jmp     word word RM_Default_INT_Routine
DB      6AH,0F1H
jmp     word RM_Default_INT_Routine
DB      6AH,0F2H
jmp     word RM_Default_INT_Routine
DB      6AH,0F3H
jmp     word RM_Default_INT_Routine
DB      6AH,0F4H
jmp     word RM_Default_INT_Routine
DB      6AH,0F5H
jmp     word RM_Default_INT_Routine
```

```

DB      6AH,0F6H
jmp     word RM_Default_INT_Routine
DB      6AH,0F7H
jmp     word RM_Default_INT_Routine
DB      6AH,0F8H
jmp     word RM_Default_INT_Routine
DB      6AH,0F9H
jmp     word RM_Default_INT_Routine
DB      6AH,0FAH
jmp     word RM_Default_INT_Routine
DB      6AH,0FBH
jmp     word RM_Default_INT_Routine
DB      6AH,0FCH
jmp     word RM_Default_INT_Routine
DB      6AH,0FDH
jmp     word RM_Default_INT_Routine
DB      6AH,0FEH
jmp     word RM_Default_INT_Routine
IRET                                ;For 0FFH just return. (Noise on INTA line?)

```

```

RM_Default_INT_Routine:            ;Unless told otherwise, all the above Ints will come here.

PUSH    AX                        ;Save only what will be changed. (Note. IRET has already saves the flags)
PUSH    BX
PUSH    BP

```



```

ADD    SP,2                ;Balance up stack,return
HLT                    ;Halt for all FAULTS (does not matter Stack is messed up, we will reset CPU)

```

```

RM_NMI_INT_Routine:        ;NMI Trap  (INT#2)

```

```

    PUSH    AX
    PUSH    BX
    MOV     BX,NMI_FAULT_MSG
    JMP     INT_INFO_DONE

```

```

RM_Overflow_INT_Routine:   ;Int #4, Overflow TRAP

```

```

    PUSH    AX
    PUSH    BX
    MOV     BX,OVERFLOW_ERR_MSG
    JMP     INT_INFO_DONE

```

```

RM_Bounds_INT_Routine:    ;Int #5, Bounds Check, (Return address to error, so ABORT)

```

```

    PUSH    AX
    PUSH    BX
    MOV     BX,BOUNDS_ERR_MSG
    JMP     INT_HALT

```

```

RM_Opcode_INT_Routine:    ;Int #6, Invalid Opcode, (Return address to error, so ABORT)

```

```

    PUSH    AX
    PUSH    BX
    MOV     BX,INVALID_ERR_MSG

```

```
JMP    INT_HALT
```

```
RM_Device_INT_Routine:                ;Int #7, Math Coprocessor not available, (Return address to error, so ABORT)
```

```
PUSH    AX
```

```
PUSH    BX
```

```
MOV     BX,DEVICE_ERR_MSG
```

```
JMP     INT_HALT
```

```
RM_DFault_INT_Routine:                ;Int #8, Double, (Return address to error, so ABORT)
```

```
PUSH    AX
```

```
PUSH    BX
```

```
MOV     BX,DOUBLE_ERR_MSG
```

```
JMP     INT_HALT
```

```
RM_MathSeg_INT_Routine:                ;Int #9, No Coprocessor, (ABORT anyway)
```

```
PUSH    AX
```

```
PUSH    BX
```

```
MOV     BX,COPROCESSOR_ERR_MSG
```

```
JMP     INT_HALT
```

```
RM_TSS_INT_Routine:                    ;Int #10, Invalid TSS
```

```
PUSH    AX
```

```
PUSH    BX
```

```
MOV     BX,INVALID_TSS_ERR_MSG
```

```
JMP     INT_HALT
```

```
RM_Segment_INT_Routine:           ;Int #11, Segment not present

    PUSH    AX

    PUSH    BX

    MOV     BX, SEGMENT_ERR_MSG

    JMP     INT_HALT


RM_Stack_INT_Routine:             ;Int #12, Stack Exception

    PUSH    AX

    PUSH    BX

    MOV     BX, STACK_ERR_MSG

    JMP     INT_HALT


RM_General_INT_Routine:           ;Int #13, General protection error (+ ERROR #)

    PUSH    AX

    PUSH    BX

    MOV     BX, GENERAL_ERR_MSG

    JMP     INT_HALT


RM_Page_INT_Routine:             ;Int #14, Page fault

    PUSH    AX

    PUSH    BX

    MOV     BX, PAGE_ERR_MSG

    JMP     INT_INFO_DONE
```



```

RM_Intel_INT_Routine:                ;Int #15,  Intel reserved Int

    PUSH    AX

    PUSH    BX

    MOV     BX,RESERVED_ERR_MSG

    JMP     INT_INFO_DONE


RM_Coprocessor_INT_Routine:          ;Int #16 Cprocessor Error

    PUSH    AX

    PUSH    BX

    MOV     BX,COPROCESSOR_ERR_MSG

    JMP     INT_HALT


RM_CC_INT_Routine:                   ;INT #3,  Software CC Interrupt

    CALL    REGISTERS                ;Display Real Mode Registers

    PUSH    AX

    PUSH    BX

    PUSH    CX

    PUSH    BP

    MOV     BP,SP

    MOV     BX,IP_ADDRESS_MSG        ;IP=

    CALL    PRINT_STRING

    MOV     AX,[BP+12]               ;Get return IP address on stack, SEGMENT

```

```

CALL AX_HEXOUT
MOV CL, ':'
CALL CO
MOV AX, [BP+10] ;Get return IP address on stack, OFFSET
CALL AX_HEXOUT
MOV BX, H_MSG_CRLF
CALL PRINT_STRING
POP BP
POP CX
POP BX
POP AX
ADD SP, 2 ;Balance up stack, return
IRET ;Remember the byte saved on the stack is extended to a word by the CPU

```

```

RM_TRACE_INT_Routine: ;INT#1, trace mode
CALL REGISTERS ;Display Real Mode Registers
PUSH AX
PUSH BX
PUSH CX
PUSH BP
MOV BP, SP
MOV BX, IP_ADDRESS_MSG ;IP=
CALL PRINT_STRING
MOV AX, [BP+12] ;Get return IP address on stack, SEGMENT

```

```

CALL AX_HEXOUT
MOV CL, ':'
CALL CO
MOV AX, [BP+10] ;Get return IP address on stack, OFFSET
CALL AX_HEXOUT
MOV BX, H_MSG_CRLF
CALL PRINT_STRING
POP BP
POP CX
POP BX
POP AX
ADD SP, 2 ;Balance up stack, return
IRET

```

```

TRACE_MODE_ON: ;Turn on CPU trace mode
PUSH AX
PUSHF ;EFLAGS to stack
POP AX
OR AX, 0100H ;Turn ON trace Bit
PUSH AX
POPF ;Move back into EFLAGS
POP AX
RET

```

```
TRACE_MODE_OFF:                                ;Turn off CPU trace mode
```

```
    PUSH    AX
```

```
    PUSHF                                ;EFLAGS to stack
```

```
    POP     AX
```

```
    AND     AX,0FEFFH                    ;Turn OFF trace Bit
```

```
    PUSH    AX
```

```
    POPF                                ;Move back into EFLAGS
```

```
    POP     AX
```

```
    RET
```

```
;----- Run diagnostic software Interrupt tests -----
```

```
;>>>>>>> Make sure the Timer interrupt (VI0) is not active.    <<<<<<<<<<
```

```
;>>>>>>> Remove jumper if necessary on board                <<<<<<<<<<
```

```
;
```

```
;We will fill out all 256 Interrupt vectors with a diagnostic routine to show
```

```
;what interrupt was triggered. A menu with a few samples is provided.
```

```
;
```

```
SOFT_INTS:
```

```
    mov     bx,INT_SIGNON                ;Send a signon message
```

```
    call    PRINT_STRING
```

```
    CLD                                  ;Default to direction up
```

```
    CLI                                  ;No hardware Ints
```

```

mov     ax,cs                ;Note this is just a simplified sub-section of the SETUP_IBM_BIOS routine
mov     ds,ax                ;DS is this ROM's CS

CALL    SETUP_INT_TABLE      ;Fill all RM 8086 interrupts initially with a default error trapping pointer

CALL    GET2DIGITS           ;Input value to AL

CMP     AH,ESC                ;Was an escape requested
JZ      INTS_DONE

CMP     AL,0
JZ      DIVIDE_CHECK
CMP     AL,4
JZ      OVERFLOW_CHECK
CMP     AL,6
JZ      INVALID_CHECK
CMP     AL,0DH
JZ      BAD_GP_FAULT_CHECK
CMP     AL,40H
JZ      BAD_SOFT_40_CHECK
CMP     AL,0F0H
JZ      BAD_SOFT_F0_CHECK
JMP     NOT_FAULTS

```

DIVIDE_CHECK:

```
    CALL    CRLF
    MOV     AX,0FFFFH
    MOV     BX,0
    DIV     BX                ;Try divide by 0
    CALL    CRLF
    JMP     SOFT_INTS        ;Normally will not get here
```

OVERFLOW_CHECK:

```
    CALL    CRLF
    INT     4
    CALL    CRLF
    JMP     SOFT_INTS
```

INVALID_CHECK:

```
    CALL    CRLF
    INT     6
    CALL    CRLF
    JMP     SOFT_INTS
```

BAD_GP_FAULT_CHECK:

```
    CALL    CRLF
    INT     0DH
    CALL    CRLF
    JMP     SOFT_INTS
```

BAD_SOFT_40_CHECK:

```
    CALL    CRLF
    INT     40H
```

```

        CALL    CRLF
        JMP     SOFT_INTS

BAD_SOFT_F0_CHECK:
        CALL    CRLF
        INT     0F0H
        CALL    CRLF
        JMP     SOFT_INTS

NOT_FAULTS:
        CALL    CRLF
        MOV     BX,INT_RANGE_MSG
        CALL    PRINT_STRING
        JMP     SOFT_INTS

INTS_DONE:
        RET

;----- Routine to Setup the RM Interrupt jump table in RAM (0-400H) -----

SETUP_INT_TABLE:                ;Note we assume INTS are turned off
        PUSHF
        PUSH    AX
        PUSH    BX
        PUSH    CX
        PUSH    DS

```

```

PUSH    ES

PUSH    SI

PUSH    DI


    mov     ax,cs                ;Note this is just a simplified sub-section of the SETUP_IBM_BIOS routine
    mov     ds,ax                ;DS is this ROM's CS
    sub     ax,ax
    mov     es,ax                ;ES: = 0H in RAM for STOW's below, DS: = CS:(here).
CLD                                ;Default to direction up
CLI                                ;Just in case


    mov     cx,256                ;Fill all 8086 interrupts initially with a default error trapping pointer
    sub     di,di                ;Clear destination register start at RAM 0H
    sub     bx,bx                ;Start at location 0
SETI1:  mov     ax,bx
    add     AX,RM_INT_JUMP_TABLE    ;Set to illustrate non assigned int
    stosw                ;Remember ES: is used for final location with STOSW
    mov     ax,cs                ;Interrupt segment pointer to here (always the same).
    stosw                ;<-- Note the default segment will be this CS for all ints below
    add     bx,5                ;Point to next int routine below in the 256 list
    loop    SETI1


POP     DI

POP     SI

POP     ES

```



```

POP    DS
POP    CX
POP    BX
POP    AX
POPF
RET

```

```

;----- LOAD XMODEM FILE via CONSOLE-IO PORT -----

```

```

XMODEM_LOAD:

```

```

    mov    bx,MODEM_SIGNON          ;Send Modem signon message
    call   PRINT_STRING
    CLD                               ;Default to direction up
    CLI                               ;No hardware Ints

    mov     ax,cs                    ;Just in case different
    mov     ds,ax                   ;DS is this ROM's CS

    PUSH    BP
    MOV     BP,SP                   ;Will store certain variables well below stack
    MOV     byte [BP-RECVD_SECT_NO],0
    MOV     byte [BP-SECTNO],0
    MOV     byte [BP-ERRCT],0

```

```

CALL    INIT_SCC                ;MASTER RESET THE ZILOG SCC

                                ;GOBBLE UP GARBAGE CHARS FROM THE LINE

mov     bx, RAM_DESTINATION    ;Ask for destination

call    PRINT_STRING

CALL    GET5DIGITS              ;Get (up to) 20 bit parameter. 16 bit value (4 digits) to DI.
                                ;If 5 digits, then the first digit is put in ES (highest nibble)

CMP     AL, ESC

JZ      MODEM_DONE

CALL    CRLF

MOV     CH, MODEM_RTS_DELAY    ;TIMEOUT DELAY

CALL    RECV

MOV     BX, START_POINTER      ;"Will load data starting at RAM location "

CALL    PRINT_STRING

call    SHOW_ADDRESS_ES_NOSPACE ;Show address

MOV     BX, H_MSG_CRLF

CALL    PRINT_STRING

CALL    CRLF

MOV     BX, DOWNLOAD_MSG       ;Speak "Downloading file"

CALL    SPEAK_STRING

```

```

RECV_LOOP:                ;<<< MAIN RECIEVE LOOP

    XOR    AX,AX           ;GET 0

    MOV     [BP-ERRCT],AL   ;Start error count with 0

RECV_HDR:

    PUSH    BX

    MOV     BX,RMSG        ;Reading sector message

    CALL    PRINT_STRING

    MOV     AL,[BP-SECTNO]

    INC     AL

    CALL    AL_HEXOUT

    MOV     BX,RAM_MSG      ;"H.  IF OK, will write to RAM location"

    CALL    PRINT_STRING

    call    SHOW_ADDRESS_ES_NOSPACE ;Show address

    MOV     BX,H_MSG

    CALL    PRINT_STRING

    POP     BX

    MOV     CH,(20*MODEM_RTS_DELAY) ;20 SEC TIMEOUT

    CALL    RECV

    JNB     RHNT0          ;WE ARE OK, NO TIMEOUT

RECV_HDR_TIMEOUT:

    MOV     BX,TOUTM        ;PRINT TIMEOUT MESSAGE

    CALL    PRINT_STRING

    MOV     AL,[BP-ERRCT]

```

```
CALL  AL_HEXOUT          ;FALL INTO CR/LF
```

```
CALL  CRLF
```

```
RCV_SECT_ERR:           ;PURGE THE LINE OF INPUT CHARS
```

```
MOV    CH,MODEM_RTS_DELAY ;~1 SEC W/NO CHARS
```

```
CALL  RCV
```

```
JNB    RCV_SECT_ERR      ;LOOP UNTIL SENDER DONE
```

```
MOV    AL,NAK
```

```
CALL  SEND              ;SEND NAK
```

```
MOV    AL,[BP-ERRCT]     ;Inc Error Count (ERRCT)
```

```
INC    AL
```

```
MOV    [BP-ERRCT],AL
```

```
CMP    AL,MODEM_ERR_LIMIT ;Currently set for 5 trys
```

```
JB     RCV_HDR           ;Go try again
```

```
CALL  CHECK_FOR_QUIT
```

```
JZ     RCV_HDR           ;Try again
```

```
MOV    BX,BAD_HEADER     ;'Unable to get a valid file header!'
```

```
CALL  PRINT_STRING
```

```
JMP    MODEM_DONE        ;Abort back to Monitor start
```

```
RHNT0: CMP    AL,SOH      ;GOT CHAR - MUST BE SOH
```

```
JZ     GOT_SOH
```

```
OR     AL,AL             ;00 FROM SPEED CHECK?
```

```
JNZ    L_2
```

```
JMP    RCV_HDR
```

```

L_2:  CMP    AL,EOT
      JNZ    L_3
      JMP    GOT_EOT
L_3:  CALL   AL_HEXOUT
      MOV    BX,ERRSOH          ;'H Received',CR,LF,'Did not get Correct SOH'
      CALL   PRINT_STRING
      JMP    RECV_SECT_ERR

```

```

GOT_SOH:                                ;We got correct SOH so now get data
      MOV    CH,MODEM_RTS_DELAY
      CALL   RECV
      JB     RECV_HDR_TIMEOUT
      MOV    DH,AL              ;D=BLK #
      MOV    CH,MODEM_RTS_DELAY
      CALL   RECV              ;GET CMA'D SECT #
      JB     RECV_HDR_TIMEOUT
      NOT    AL
      CMP    AL,DH              ;GOOD SECTOR #?
      JZ     RECV_SECTOR
                                ;GOT BAD SECTOR #
      MOV    BX,MODEM_ERR2      ;'++BAD SECTOR # IN HDR'
      CALL   PRINT_STRING
      JMP    RECV_SECT_ERR

```

```

RECV_SECTOR:                            ;Now get 128 Bytes

```

```

MOV     AL,DH                ;GET SECTOR #

MOV     [BP-RECVD_SECT_NO],AL

MOV     CL,0                 ;INIT CKSUM

MOV     BL,80H               ;128 Byte sectors always

RECV_CHAR:

MOV     CH,MODEM_RTS_DELAY   ;~1 SEC TIMEOUT

CALL    RECV                 ;GET CHAR

JNB     MODL_4

JMP     RECV_HDR_TIMEOUT

MODL_4:  MOV     [ES:DI],AL     ;<<< STORE CHAR >>>

INC     DI

DEC     BL                   ;128 Bytes done yet?

JNZ     RECV_CHAR

                                ;NEXT VERIFY CHECKSUM

MOV     DH,CL                ;SAVE CHECKSUM

MOV     CH,MODEM_RTS_DELAY   ;TIMEOUT

CALL    RECV                 ;GET CHECKSUM

JNB     MODL_5

JMP     RECV_HDR_TIMEOUT

MODL_5:

CMP     AL,DH                ;CHECK

JNZ     RECV_CKSUM_ERR

MOV     AL,[BP-RECVD_SECT_NO] ;GOT A SECTOR, WRITE IF = 1+PREV SECTOR

MOV     CH,AL                ;SAVE IT

```

```

MOV     AL,[BP-SECTNO]           ;GET PREV
INC     AL                       ;CALC NEXT SECTOR #
CMP     AL,CH                    ;MATCH?
JNZ     DO_ACK

MOV     AL,[BP-RECV_SECT_NO]     ;Indicate we transferred a sector
MOV     [BP-SECTNO],AL           ;UPDATE SECTOR #
DO_ACK:  MOV     AL,ACK
CALL    SEND
JMP     RECV_LOOP

RECV_CKSUM_ERR:
MOV     BX,MODEM_ERR3
CALL    PRINT_STRING
JMP     RECV_SECT_ERR

GOT_EOT:                               ;DONE - CLOSE UP SHOP
MOV     AL,ACK                   ;ACK THE EOT
CALL    SEND
CALL    CRLF
MOV     BX,FINISH_MSG            ;Speak downloading finished
CALL    SPEAK_STRING
MOV     BX,TRANS_DONE
EXIT2:  CALL    PRINT_STRING

```

```

MODEM_DONE:
    XOR    AL,AL
    POP    BP                ;RESTORE IT
    RET

EXIT1: MOV    BX,ABORT_MSG
        JMP    EXIT2

                                ;INITITIALIZE THE ZILOG SCC SERIAL B PORT

INIT_SCC:
    MOV    BX,SSC_MSG_INIT    ;Say Initilizing ACIA/SCC
    CALL   PRINT_STRING
    MOV    CH,14              ;Byte count (14), for below
    MOV    BX,SCCINIT         ;Table of Zilog SCC Initilization values
SCC_1: MOV    AL,[CS:BX]
    OUT    MODEM_CTL_PORT,AL  ;Program the SCC Channel B (A1,A3 or 10,12H) for 19K Baud
    INC    BX
    DEC    CH
    JNZ    SCC_1
    MOV    BX,SPEED_MSG       ;Speak, baud rate set
    CALL   SPEAK_STRING
    RET

                                ;---- SERIAL PORT GET CHARACTER ROUTINE ----

RECV:  PUSH    DX                ;SAVE

```



```

MOV     AL,5H                ;Lower RTS line

OUT     MODEM_CTL_PORT,AL    ;Sel Reg 5

MOV     AL,11101010B        ;EAH

OUT     MODEM_CTL_PORT,AL

NOP

NOP

MSEC:   MOV     DX,8000H      ;~0.1 SEC DCR COUNT

MWTI:   IN      AL,MODEM_CTL_PORT

        AND     AL,MODEM_RECV_MASK

        CMP     AL,RECV_READY

        JZ      MCHAR        ;GOT CHAR

        DEC     DX            ;COUNT DOWN

        JNZ     MWTI         ;FOR TIMEOUT

        DEC     CH            ;DCR # OF SECONDS

        JNZ     MSEC         ;MODEM TIMED OUT RECEIVING

        POP     DX            ;RESTORE DX

        STC                     ;CARRY SHOWS TIMEOUT

        RET

MCHAR:  IN      AL,MODEM_DATA_PORT

        POP     DX            ;RESTORE DE

        LAHF                     ;CALC CHECKSUM

        XCHG    AL,AH

        PUSH    AX

        XCHG    AL,AH

```

```

    ADD     AL,CL
    MOV     CL,AL
    POP     AX
    XCHG    AL,AH
    OR      AL,AL          ;TURN OFF CARRY TO SHOW NO TIMEOUT
    RET

                                ;---- SERIAL PORT SEND CHARACTER ROUTINE ----
SEND:  LAHF                                ;CHECK IF MONITORING OUTPUT
    XCHG    AL,AH
    PUSH    AX
    XCHG    AL,AH
    ADD     AL,CL          ;CALC CKSUM
    MOV     CL,AL
SENDW: IN     AL,MODEM_CTL_PORT  ;Don't worry PC is always fast enough
    AND     AL,MODEM_SEND_MASK
    CMP     AL,SEND_READY
    JNZ     SENDW
    POP     AX              ;GET CHAR
    XCHG    AL,AH
    SAHF
    OUT     MODEM_DATA_PORT,AL  ;Raise RTS line to prevent the next character arriving
    MOV     AL,5H              ;while the CPU is busy processing info
    OUT     MODEM_CTL_PORT,AL   ;Sel Reg 5
    MOV     AL,11101000B       ;E8H
    OUT     MODEM_CTL_PORT,AL

```

```
RET
```

```
CHECK_FOR_QUIT:                ;MULTIPLE ERRORS, ASK IF TIME TO QUIT
```

```
    XOR    AL,AL                ;GET 0
```

```
    MOV    [BP-ERRCT],AL        ;RESET ERROR COUNT
```

```
    MOV    BX,QUITM
```

```
    CALL   PRINT_STRING
```

```
    CALL   CICO
```

```
    PUSH   AX
```

```
    CALL   CRLF
```

```
    POP    AX
```

```
    CMP    AL,'R'
```

```
    JZ     DONE_CHECK
```

```
    CMP    AL,'Q'
```

```
    JZ     NOT_DONE_CHECK
```

```
    CMP    AL,ESC
```

```
    JZ     NOT_DONE_CHECK
```

```
    JMP    CHECK_FOR_QUIT
```

```
NOT_DONE_CHECK:
```

```
    OR     AL,AL                ;TURN OFF ZERO FLAG
```

```
DONE_CHECK:
```

```
RET
```

```

;*****
;
;    Module to Test and diagnose the www.S100Computers.com IDE Board
;
;    Instead of using the CPM86 style DS:[BX] format, we will use SS:[BP] so the the buffers
;    can reside at the top segment of available RAM. Normally this will be F000:7000H but the monitor
;    will not assume the full 1MG address space is available.
;    See the monitor initialization section where BP is setup.
;
;
;*****

MYIDE: MOV     BP,DISPLAY_FLAG           ;Do we have detail sector data display flag on or off
        MOV     AL,0FFH                 ;Set default to detailed sector display
        MOV     [BP],AL

        MOV     BX,IDE_HARDWARE         ;"Initilizing IDE Drive hardware"
        CALL    PRINT_STRING

        CALL    CLEAR$ID$BUFFER         ;Clear ID Buffer

```

```

CALL  SET_DRIVE_A      ;Select the first Drive/CF card
CALL  IDEinit          ;Initialize the board and drive #0. If there is no drive abort
JZ    INIT1_OK

MOV   BX,INIT_1_ERROR
CALL  PRINT_STRING
JMP   INIT

```

INIT1_OK:

```

CALL  CLEAR$ID$BUFFER      ;Clear ID Buffer

CALL  SET_DRIVE_B      ;Select the second Drive/CF card (Do not mess with CPM Drive 0)
CALL  IDEinit          ;Initialize drive #1. If there is no drive abort
JZ    INIT2_OK

CALL  CLEAR$ID$BUFFER      ;Clear ID Buffer

MOV   BX,INIT_2_ERROR      ;Warn second IDE drive did not initilize
CALL  PRINT_STRING        ;Since first drive was OK we will still go to INIT2_OK

```

INIT2_OK:

```

CALL  SET_DRIVE_A      ;Back to first drive/CF Card

CALL  DRIVE_ID          ;Get the drive 0 ID info. If there is no drive just abort

```

JZ INIT3_OK

MOV BX,BAD_DRIVE ;"Error obtaining the Drive ID"

CALL PRINT_STRING

JMP INIT

INIT3_OK: ;Check we have a valid IDE drive

MOV BP,(IDE_Buffer+12) ;Note always SS: = CS:

MOV AX,[BP]

OR AX,AX ;If there are zero sectors then something wrong

JNZ INIT4_OK

MOV BX,BAD_DRIVE ;"Error obtaining first Drive ID"

CALL PRINT_STRING

JMP INIT

INIT4_OK:

MOV BP,RAM_DMA ;Set default position will be first sector block

MOV word[BP],IDE_Buffer ;DMA always initially to IDE_Buffer,

MOV BP,RAM_SEC

MOV word[BP],0H ;Sec 0

MOV BP,RAM_TRK

MOV word[BP],0H ;Track 0

```

CALL    IDEinit                ;For some reason this need to be here after getting the drive ID.
                                ;otherwise sector #'s are off by one! (Probably because on non-LBA reads)

CALL    WR_LBA                 ;Update LBA on "1st" drive

```

```

;----- MAIN IDE DRIVE DIAGNOSTIC MENU -----

```

```

IDE_LOOP:

```

```

    MOV    AX,CS                ;Just in case somehow they changed somewhere below

```

```

    MOV    DS,AX

```

```

    MOV    ES,AX

```

```

    MOV    BX,IDE_SIGNON0       ;List IDE command options

```

```

    CALL   PRINT_STRING

```

```

    MOV    BP,CURRENT_IDE_DRIVE

```

```

    MOV    AL,[BP]

```

```

    OR     AL,AL

```

```

    JNZ    SIGN_B

```

```

    MOV    BX,CURRENT_MSG_A

```

```

    JMP    IDE_LOOP0

```

```

SIGN_B:    MOV    BX,CURRENT_MSG_B

```

```

IDE_LOOP0:

```

```

CALL    PRINT_STRING

MOV     BX,IDE_SIGNON4           ;List IDE command options
CALL    PRINT_STRING

MOV     BP,DISPLAY_FLAG         ;Do we have detail sector data display flag ON or OFF
MOV     AL,[BP]                  ;NZ = on
OR      AL,AL
JNZ     IDE_LOOP1
MOV     BX,IDE_SIGNON1          ;"ON"
JMP     IDE_LOOP2

IDE_LOOP1:
MOV     BX,IDE_SIGNON2          ;"OFF"

IDE_LOOP2:
CALL    PRINT_STRING

MOV     BX,IDE_SIGNON3          ;List IDE command options
CALL    PRINT_STRING

CALL    DISPLAY_POSITION        ;Display current Track,sector,head#

CALL    CRLF

MOV     BX,IDE_MENU             ;Enter a command
CALL    PRINT_STRING

call    CICO                    ;Get a command from Console

```



```

mov     ah,0

CMP     AL,ESC           ;Abort if ESC

JNZ     NOT_ESC

JMP     INIT             ;Back to start of Monitor

NOT_ESC:cmp  al,'A'       ;Find menu option from table

        jb     IDE_LOOP   ;must be A to Z

        cmp    al,'Z'

        jg     IDE_LOOP

        sub    al,'A'      ;calculate offset

        shl    al,1        ;X 2

        add    ax,IDE_TABLE ;Note DS:=CS:

        mov    bx,ax

        CALL   CRLF

        mov    ax,[cs:bx]  ;get location of routine CS:[BX]

        call   ax          ;<----- This is the IDE Menu CMD call

        jmp    IDE_LOOP   ;finished

;

        INDIVIDUAL IDE DRIVE MENU COMMANDS

;-----Select Drive/CF card -----

SET_DRIVE_A:                ;Select First Drive

        MOV     AL,0

```

SELECT_DRIVE:

```

    MOV     BP,CURRENT_IDE_DRIVE

    MOV     [BP],AL

    OUT     IDEDrivePort,AL           ;Select Drive 0 or 1

    RET

```

SET_DRIVE_B: ;Select Drive 1

```

    MOV     AL,1

    JMP     SELECT_DRIVE

```

;----- Do the IDeNtify drive command, and display the IDE_Buffer -----

DRIVE_ID:

```

    CALL    IDEwaitnotbusy

    JNB     L_5

    XOR     AX,AX

    DEC     AX                       ;NZ if error

    RET                                           ;If Busy return NZ

```

L_5: MOV DH,COMMANDid

MOV DL,REGcommand

CALL IDEwr8D ;issue the command

CALL IDEwaitdrq ;Wait for Busy=0, DRQ=1

JNB L_6

```

        JMP     SHOWErrors

L_6:    MOV     CH,0                ;256 words
        MOV     BP,IDE_Buffer      ;Store data here (remember CS: = SS:)
        CALL    MoreRD16           ;Get 256 words of data from REGdata port to ss:[BP]


        MOV     BX,msgmdl          ;print the drive's model number
        CALL    PRINT_STRING
        MOV     BP,(IDE_Buffer + 54)
        MOV     CH,10              ;Character count in words
        CALL    Print_ID_Info      ;Print [HL], [B] X 2 characters
        CALL    CRLF

                                ; print the drive's serial number

        MOV     BX,msgsn
        CALL    PRINT_STRING
        MOV     BP,(IDE_Buffer + 20)
        MOV     CH,5              ;Character count in words
        CALL    Print_ID_Info
        CALL    CRLF

                                ;PRINT_STRING the drive's firmware revision string

        MOV     BX,msgrev
        CALL    PRINT_STRING
        MOV     BP,(IDE_Buffer + 46)
        MOV     CH,2

```

```

CALL  Print_ID_Info      ;Character count in words
CALL  CRLF

                                ;print the drive's cylinder, head, and sector specs

MOV   BX,msgcy
CALL  PRINT_STRING
MOV   BP,(IDE_Buffer + 2)
CALL  Print_ID_HEX
MOV   BX,msghd
CALL  PRINT_STRING
MOV   BP,(IDE_Buffer + 6)
CALL  Print_ID_HEX
MOV   BX,msgsc
CALL  PRINT_STRING
MOV   BP,(IDE_Buffer + 12)      ;Sectors/track
CALL  Print_ID_HEX
CALL  CRLF
XOR   AX,AX                    ;Ret Z
RET

```

; Print a string located [BP] (Used only by the above DISK ID routine)

Print_ID_Info:

```

MOV   CL,[BP+1]              ;Text is low byte high byte format
CALL  CO
MOV   CL,[BP]

```

```

CALL    CO
INC     BP
INC     BP
DEC     CH
JNZ     Print_ID_Info
RET

```

; Print a 16 bit number, located [BP] (Used only by the above DISK ID routine)

; (Note Special Low Byte First. Used only for Drive ID)

Print_ID_HEX:

```

MOV     AL,[BP+1]           ;Index to high byte first
CALL    AL_HEXOUT
MOV     AL,[BP]             ;Now low byte
CALL    AL_HEXOUT
RET

```

;----- Read the current selected sector (based on LBA) to the IDE Buffer

READ_SEC:

```

MOV     AX,CS
MOV     DS,AX
MOV     BP,RAM_DMA
MOV     word [BP],IDE_Buffer ;DMA initially to IDE_Buffer

CALL    READSECTOR

```

```

JZ      Main1B

CALL    CRLF                ;Here if there was a problem

RET

```

```

Main1B:  MOV     BX,msgrd          ;Sector read OK

CALL    PRINT_STRING

MOV     BP,DISPLAY_FLAG        ;Do we have detail sector data display flag on or off
MOV     AL,[BP]                ;NZ = on
OR      AL,AL
JNZ     SHOW_SEC_RDATA
RET

```

```

SHOW_SEC_RDATA:
MOV     BP,RAM_DMA
MOV     word [BP],IDE_Buffer    ;DMA initially to IDE_Buffer
CALL    DISPLAY_SEC
MOV     BX,CR_To_Continue
CALL    PRINT_STRING
CALL    CI
RET

```

;----- Write the current selected sector (based on LBA) from the IDE Buffer

WRITE_SEC:

```

    MOV     AX,CS
    MOV     DS,AX

    MOV     BX,CONFIRM_WR_MSG    ;Are you sure?
    CALL    PRINT_STRING
    CALL    CICO
    CMP     AL,'Y'
    JZ      WR_SEC_OK1
    CALL    CRLF                  ;Here if there was a problem
    RET

```

WR_SEC_OK1:

```

    MOV     BP,RAM_DMA
    MOV     word [BP],IDE_Buffer    ;DMA initially to IDE_Buffer

    CALL    WRITESECTOR            ;Will write whatever is in the IDE_Buffer

    JZ      Main2B
    CALL    CRLF                  ;Here if there was a problem
    RET

```

Main2B: MOV BX,msgrd ;Sector written OK

```

    CALL    PRINT_STRING

```

```

MOV     BP,DISPLAY_FLAG           ;Do we have detail sector data display flag on or off
MOV     AL,[BP]                   ;NZ = on
OR      AL,AL
JNZ     SHOW_SEC_WDATA
RET

```

SHOW_SEC_WDATA:

```

MOV     BP,RAM_DMA
MOV     word [BP],IDE_Buffer       ;DMA initially to IDE_Buffer
CALL    DISPLAY_SEC
MOV     BX,CR_To_Continue
CALL    PRINT_STRING
CALL    CI
RET

```

;----- Set a new LBA value from imputted Track/Sec info. Send to drive

```

SET_LBA:MOV  AX,CS
MOV        DS,AX
MOV        BX,SET_LBA_MSG          ;Set new LBA and send to drive
CALL       PRINT_STRING
CALL       GEN_HEX32_LBA           ;Get new CPM style Track & Sector number and put them in RAM at RAM_SEC & RAM_TRK
JB         main3b                  ;Ret C set if abort/error
CALL       WR_LBA                  ;Update LBA on drive

```



```

main3b:      CALL    CRLF

             RET

;----- Toggle detailed sector display on/off

DISPLAY:

    MOV      AX,CS

    MOV      DS,AX

    MOV      BP,DISPLAY_FLAG      ;Do we have detail sector data display flag on or off

    MOV      AL,[BP]              ;NZ = on

    NOT      AL

    MOV      [BP],AL

    RET

;----- Point current sector to next sector

NEXT_SECT:

    CALL     GET_NEXT_SECT

    JNZ      AT_END

    RET

AT_END:

    MOV      BX,AT_END_MSG        ;Tell us we are at end of disk

    CALL     PRINT_STRING

    RET

```

```
;----- Point current sector to previous sector
```

```
PREV_SECT:
```

```
    CALL    GET_PREV_SECT
```

```
    JNZ     AT_START
```

```
    RET
```

```
AT_START:
```

```
    MOV     BX,AT_START_MSG           ;Tell us we are at start of disk
```

```
    CALL    PRINT_STRING
```

```
    RET
```

```
;----- Sequentially read sectors from disk starting at current LBA position
```

```
SEQ_SEC_RD:
```

```
    MOV     AX,CS
```

```
    MOV     DS,AX
```

```
    CALL    IDEwaitnotbusy
```

```
    JNB     MORE_SEC
```

```
    JMP     SHOWerrors
```

```
MORE_SEC:
```

```
    CALL    CRLF
```

```
    MOV     BP,RAM_DMA               ;Set DMA initially to IDE_Buffer
```

```
    MOV     CL,'<'
```

```
    CALL    CO
```

```
MOV     AX,BP
CALL    AX_HEXOUT
```

```
MOV     word [BP],IDE_Buffer
MOV     CL, '.'
CALL    CO
MOV     AX,[BP]
CALL    AX_HEXOUT
MOV     CL, '>'
CALL    CO
```

```
CALL    READSECTOR           ;If there are errors they will show up in READSECTOR
JZ      SEQOK
```

```
MOV     BX,CONTINUE_MSG      ;If an error ask if we wish to continue
CALL    PRINT_STRING
CALL    CICO
CMP     AL,ESC               ;Abort if ESC
JNZ     SEQOK
RET
```

```
SEQOK: CALL    DISPLAY_POSITION ;Display current Track,sector,head#
```

```
MOV     BP,DISPLAY_FLAG      ;Do we have detail sector data display flag on or off
MOV     AL,[BP]              ;NZ = on
```

```

OR      AL,AL

JZ      MORES2

MOV     BP, RAM_DMA           ;Point DMA to IDE_Buffer again

MOV     word [BP], IDE_Buffer

CALL    DISPLAY_SEC

MORS2:  CALL    CSTS           ;Any keyboard character will stop display

JZ      NO_WAIT

CALL    CI

MOV     BX, CONTINUE_MSG

CALL    PRINT_STRING

CALL    CI

CMP     AL, ESC

JNZ     NO_WAIT

RET                                           ;Bug, is returning to monitor, must be a stack problem!

NO_WAIT:CALL GET_NEXT_SECT       ;Point LBA to next sector

JZ      MORE_SEC               ;Note will go to last sec on disk unless stopped

RET

;----- Read N Sectors to disk

;Note unlike the normal sector read, this routine increments the DMA address after each sector read

```

N_RD_SEC:

```

MOV     AX,CS
MOV     DS,AX
MOV     BX,READN_MSG
CALL    PRINT_STRING
CALL    GET2DIGITS           ;Hex to AL

MOV     BP,SECTOR_COUNT     ;store sector count
MOV     [BP],AL

MOV     BP,RAM_DMA_STORE
MOV     word [BP],IDE_Buffer ;DMA_STORE initially to IDE_Buffer

```

NextRSec:

```

MOV     BX, READN_S_MSG
CALL    PRINT_STRING
CALL    WR_LBA              ;Update LBA on drive
CALL    DISPLAY_POSITION    ;Display current Track,sector,head#

MOV     BP,RAM_DMA_STORE
MOV     AX,[BP]             ;Get last value of DMA address
MOV     BP,RAM_DMA
MOV     [BP],AX             ;Store it in DMA address

```

```
CALL    READSECTOR          ;Actully, Sector/track values are already updated
```

```
MOV     BP, RAM_DMA
```

```
MOV     AX, [BP]            ;Store it in DMA_STORE address
```

```
MOV     BP, RAM_DMA_STORE
```

```
MOV     [BP], AX
```

```
MOV     BP, SECTOR_COUNT
```

```
MOV     AL, [BP]
```

```
DEC     AL
```

```
MOV     [BP], AL
```

```
JNZ     NEXT_SEC_NRD
```

```
RET
```

```
NEXT_SEC_NRD:
```

```
CALL    GET_NEXT_SECT
```

```
JZ      NextRSec
```

```
MOV     BX, AT_END_MSG      ;Tell us we are at end of disk
```

```
CALL    PRINT_STRING
```

```
RET
```

```
;----- Write N Sectors to disk
```

;Note unlike the normal sector write routine, this routine increments the DMA address after each write.

N_WR_SEC:

```

    MOV     AX,CS
    MOV     DS,AX
    MOV     BX,CONFIRM_WR_MSG    ;Are you sure?
    CALL    PRINT_STRING
    CALL    CICO
    CMP     AL,'Y'
    JZ      WR_SEC_OK2
    CALL    CRLF                  ;Here if there was a problem
    RET

```

WR_SEC_OK2:

```

    MOV     BX,WRITEN_MSG
    CALL    PRINT_STRING
    CALL    GET2DIGITS            ;Hex to AL

    MOV     BP,SECTOR_COUNT      ;store sector count
    MOV     [BP],AL

    MOV     BP,RAM_DMA_STORE
    MOV     word [BP],IDE_Buffer ;DMA_STORE initially to IDE_Buffer

```

NextWSec:

```

MOV     BX, WRITEN_S_MSG
CALL    PRINT_STRING
CALL    WR_LBA           ;Update LBA on drive
CALL    DISPLAY_POSITION ;Display current Track,sector,head#

MOV     BP, RAM_DMA_STORE

MOV     AX, [BP]          ;Get last value of DMA address
MOV     BP, RAM_DMA
MOV     [BP], AX          ;Store it in DMA address

CALL    WRITESECTOR      ;Actully, Sector/track values are already updated

MOV     BP, RAM_DMA
MOV     AX, [BP]          ;Store it in DMA_STORE address
MOV     BP, RAM_DMA_STORE
MOV     [BP], AX

MOV     BP, SECTOR_COUNT
MOV     AL, [BP]
DEC     AL
MOV     [BP], AL
JNZ     NEXT_SEC_NWR
RET

NEXT_SEC_NWR:
CALL    GET_NEXT_SECT

```



```

JZ      NextWSec

MOV     BX,AT_END_MSG      ;Tell us we are at end of disk

CALL    PRINT_STRING

RET

```

```

;----- Format current disk

```

```

FORMAT:

```

```

MOV     AX,CS
MOV     DS,AX
MOV     BP,CURRENT_IDE_DRIVE
MOV     AL,[BP]
OR      AL,AL
JNZ     FORM_B
MOV     BX,FORMAT_MSG_A
JMP     FORM_X

```

```

FORM_B:  MOV     BX,FORMAT_MSG_B

```

```

FORM_X: CALL PRINT_STRING

```

```

MOV     BX, CONFIRM_WR_MSG ;Are you sure?

CALL    PRINT_STRING

CALL    CICO

CMP     AL,'Y'

JZ      FORMAT_OK

RET

```

FORMAT_OK:

```

MOV     AX,0                ;Back to CPM sector 0
MOV     BP, RAM_SEC        ;Get Current Sector
MOV     [BP],AX             ;0 to CPM Sectors

MOV     BP, RAM_TRK        ;And track
MOV     [BP],AX

MOV     AX,0E5E5H          ;First set Sector pattern to E5's
CALL    RAM_FILL
CALL    CRLF

```

NEXT_FORMAT:

```

MOV     BP, RAM_DMA        ;Point DMA to the area
MOV     word [BP],IDE_Buffer

CALL    WRITESECTOR        ;Will return error if there was one
JZ      NEXTF1             ;Z means the sector write was OK

MOV     BX,FORMAT_ERR      ;Indicate an error
CALL    PRINT_STRING
CALL    SHOW_TRACK_SEC      ;Show current location of error
CALL    CRLF
JMP     FNEXTSEC3

```

```

NEXTF1:      MOV     BP,RAM_SEC           ;Get Current Sector

             MOV     AX,[BP]

             OR      AX,AX                ;At start of each track give an update

             JNZ     FNEXTSEC2

             CALL    SHOW_TRACK

FNEXTSEC2:

             CALL    CSTS                 ;Any keyboard character will stop display

             JZ      FNEXTSEC1

             CALL    CI                   ;Flush character

FNEXTSEC3:

             MOV     BX,CONTINUE_MSG

             CALL    PRINT_STRING

             CALL    CICO

             CMP     AL,ESC

             JNZ     FNEXTSEC1

F_DONE:      MOV     AL,0                 ;Login drive A:

             CALL    SELECT_DRIVE

             MOV     BP,CURRENT_IDE_DRIVE

             MOV     [BP],AL

             RET

FNEXTSEC1:

             CALL    GET_NEXT_SECT

```

```

JZ     NEXT_FORMAT

MOV     BX,AT_END_MSG      ;Tell us we are at end of disk

CALL    PRINT_STRING

JMP     F_DONE

```

;----- Copy Drive A: to Drive B: -----

COPY_AB:

```

MOV     AX,CS
MOV     DS,AX
MOV     BX,DiskCopyMsg
CALL    PRINT_STRING
CALL    CICO
CMP     AL,'Y'
JZ      COPY_AB1
JMP     C_DONE

```

COPY_AB1:

```

MOV     BP, RAM_SEC        ;Start with CPM sector 0
MOV     AX,0
MOV     [BP],AX
MOV     BP,RAM_TRK        ;Start with CPM Track 0
MOV     AX,0
MOV     [BP],AX            ;High & Low Track to 0
CALL    CRLF
CALL    CRLF

```

NextDCopy:

```

MOV     AL,0                ;Login drive A:
CALL    SELECT_DRIVE

CALL    WR_LBA              ;Update LBA on "A:" drive

MOV     BP, RAM_DMA
MOV     word [BP], IDE_Buffer ;DMA initially to IDE_Buffer

CALL    READSECTOR         ;Get sector data from A: drive to buffer

MOV     AL,1                ;Login drive B:
CALL    SELECT_DRIVE

CALL    WR_LBA              ;Update LBA on "B:" drive

MOV     BP, RAM_DMA
MOV     word [BP], IDE_Buffer ;DMA initially to IDE_Buffer

CALL    WRITESECTOR        ;Write buffer data to sector on B: drive
JZ      COPY_OK1

MOV     BX, COPY_ERR        ;Indicate an error
CALL    PRINT_STRING

```

```

CALL  SHOW_TRACK_SEC          ;Show current location of error
CALL  CRLF
JMP    COPY_OK3

```

COPY_OK1:

```

MOV    BP, RAM_SEC            ;Get Current Sector
MOV    AX, [BP]
OR     AX, AX                  ;At start of each track give an update
JNZ    COPY_OK2

```

```

CALL  SHOW_TRACK

```

COPY_OK2:

```

CALL  CSTS                    ;Any keyboard character will stop display
JZ     C_NEXTSEC1
CALL  CI                       ;Flush character

```

COPY_OK3:

```

MOV    BX, CONTINUE_MSG
CALL  PRINT_STRING
CALL  CICO
CMP    AL, ESC
JNZ    C_NEXTSEC1

```

C_DONE: MOV AL, 0 ;Login drive A:

```

CALL  SELECT_DRIVE
MOV    BP, CURRENT_IDE_DRIVE

```

```
MOV    [BP],AL
```

```
RET
```

```
C_NEXTSEC1:
```

```
CALL   GET_NEXT_SECT      ;Update to next sector/track
```

```
JNZ    C_NEXTSEC2
```

```
JMP    NextDCopy
```

```
C_NEXTSEC2:
```

```
MOV    BX,CopyDone        ;Tell us we are all done.
```

```
CALL   PRINT_STRING
```

```
JMP    C_DONE
```

```
;----- Verify Drive A: = B: -----
```

```
VERIFY_AB:
```

```
MOV    AX,CS
```

```
MOV    DS,AX
```

```
MOV    BX,DiskVerifyMsg
```

```
CALL   PRINT_STRING
```

```
MOV    BP,RAM_SEC         ;Start with CPM sector 0
```

```
MOV    AX,0
```

```
MOV    [BP],AX
```

```
MOV    BP,RAM_TRK         ;Start with CPM Track 0
```

```

MOV     AX,0
MOV     [BP],AX                ;High & Low Track to 0

```

```
CALL    CRLF
```

```
CALL    CRLF
```

NextVCopy:

```
MOV     AL,0                  ;Login drive A:

```

```
CALL    SELECT_DRIVE

```

```
CALL    WR_LBA                ;Update LBA on "A:" drive

```

```
MOV     BP,RAM_DMA

```

```
MOV     word [BP],IDE_Buffer   ;DMA initially to IDE_Buffer

```

```
CALL    READSECTOR            ;Get sector data from A: drive to buffer

```

```
MOV     AL,1                  ;Login drive B:

```

```
CALL    SELECT_DRIVE

```

```
CALL    WR_LBA                ;Update LBA on "B:" drive

```

```
MOV     BP,RAM_DMA

```

```
MOV     word [BP],IDE_Buffer2  ;DMA initially to IDE_Buffer2

```



```

CALL    READSECTOR

MOV     DI,IDE_Buffer2
MOV     SI,IDE_Buffer
MOV     CX,512             ;Length of sector in words

NEXT_CMP:
MOV     AL, [SS:DI]        ;Note we have to use SS:
CMP     AL, [SS:SI]
JNZ     VER_ERROR
INC     DI
INC     SI
LOOP    NEXT_CMP           ;CX will contain count of words done so far, (0 if done OK)
JMP     VERIFY_OK

VER_ERROR:
MOV     BX,VERIFY_ERR      ;Indicate an error
CALL    PRINT_STRING
CALL    SHOW_TRACK_SEC     ;Show current location of error
MOV     BX,DRIVE1_MSG      ;' Drive A',CR,LF
CALL    PRINT_STRING

MOV     SI,IDE_Buffer
MOV     CX,512             ;Length of sector in words

VER_SOURCE:

```

```

MOV     AL, [SS:SI]           ;Note we have to use SS:
CALL    AL_HEXOUT
INC     SI
LOOP    VER_SOURCE
CALL    CRLF
CALL    SHOW_TRACK_SEC        ;Show current location of error
MOV     BX,DRIVE2_MSG         ;' Drive B',CR,LF
CALL    PRINT_STRING

MOV     SI,IDE_Buffer2
MOV     CX,512                ;Length of sector in words
VER_DEST:
MOV     AL, [SS:DI]           ;Note we have to use SS:
CALL    AL_HEXOUT
INC     DI
LOOP    VER_DEST
CALL    CRLF
JMP     VERIFYT               ;Do not ask for a continue message here. Just continue
                                ;If you want it change to VERIFYT1

VERIFY_OK:
MOV     BP,RAM_SEC            ;Get Current Sector
MOV     AX,[BP]
OR      AX,AX                  ;At start of each track give an update
JNZ     VERIFYT

```

```

        CALL    SHOW_TRACK

VERIFYT:CALL  CSTS                ;Any keyboard character will stop display
        JZ      V_NEXTSEC1

        CALL    CI                ;Flush character

VERIFYT1:
        MOV     BX,CONTINUE_MSG
        CALL    PRINT_STRING
        CALL    CICO
        CMP     AL,ESC
        JNZ     V_NEXTSEC1
        JMP     V_NEXTSEC3

V_NEXTSEC1:
        CALL    GET_NEXT_SECT     ;Update to next sector/track
        JNZ     V_NEXTSEC2
        JMP     NextVCopy

V_NEXTSEC2:
        MOV     BX,VerifyDone     ;Tell us we are all done.
        CALL    PRINT_STRING

V_NEXTSEC3:
        MOV     AL,0              ;Login drive A:
        CALL    SELECT_DRIVE
        MOV     BP,CURRENT_IDE_DRIVE
        MOV     [BP],AL
        RET

```

```
;----- Fill RAM buffer with 0's
```

```
RAMCLEAR:
```

```
    MOV    AX,CS
```

```
    MOV    DS,AX
```

```
    MOV    AX,0
```

```
RAM_FILL:
```

```
    MOV    BP,IDE_Buffer
```

```
    MOV    CX,256           ;512 bytes total
```

```
CLEAR1:    MOV    [BP],AX           ;Note this will be SS:BP
```

```
    INC    BP
```

```
    INC    BP
```

```
    LOOP   CLEAR1
```

```
    MOV    BX,FILL_MSG
```

```
    CALL   PRINT_STRING
```

```
    RET
```

```
;----- Power up a Hard Disk
```

```
SPINUP:      MOV     DH,COMMANDspinup
```

```
spup2: MOV     DL,REGcommand
```

```
        CALL    IDEwr8D
```

```
        CALL    IDEwaitnotbusy
```

```
        JNB     L_7
```

```
        JMP     SHOWerrors
```

```
L_7:  OR      AL,AL                ;Clear carry
```

```
        RET
```

```
;----- Tell the Hard disk to power down
```

```
SPINDOWN:
```

```
        CALL    IDEwaitnotbusy
```

```
        JNB     L_8
```

```
        JMP     SHOWerrors
```

```
L_8:  MOV     DH,COMMANDspindown
```

```
        JMP     spup2
```

```
;----- Back to parent 8086 Monitor commands
```

```
QUIT_IDE:
```

```
        JMP     INIT
```

```
;===== Support Routines FOR IDE MODULE =====
```

```
;Generate an LBA sector number with data input from CPM style Track# & Sector#
```

```
GEN_HEX32_LBA:
```

```
    MOV    BX,ENTERRAM_SECL    ;Enter sector number, low
    CALL   PRINT_STRING
    CALL   GET2DIGITS          ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)
    MOV    BP,RAM_SEC
    MOV    [BP],AL             ;Note: no check that data is < MAXSEC
    CALL   CRLF

    MOV    BX,ENTERRAM_TRKL    ;Enter low byte track number
    CALL   PRINT_STRING
    CALL   GET2DIGITS          ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)
    MOV    BP,RAM_TRK
    MOV    [BP],AL
    CALL   CRLF

    MOV    BX,ENTERRAM_TRKH    ;Enter high byte track number
    CALL   PRINT_STRING
    CALL   GET2DIGITS          ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)
    MOV    BP,RAM_TRK+1
```

```

MOV     [BP],AL
XOR     AL,AL
OR      AL,AL           ;To return NC
RET

```

```

DISPLAY_POSITION:           ;Display current track,sector & head position

```

```

MOV     BX,msgCPMTRK       ;Display in LBA format
CALL    PRINT_STRING       ;---- CPM FORMAT ----
MOV     BP,RAM_TRK+1
MOV     AL,[BP]             ;High TRK byte
CALL    AL_HEXOUT
DEC     BP
MOV     AL,[BP]             ;Low TRK byte
CALL    AL_HEXOUT

```

```

MOV     BX,msgCPMSEC
CALL    PRINT_STRING       ;SEC = (16 bits)
MOV     BP,RAM_SEC+1       ;High Sec
MOV     AL,[BP]
CALL    AL_HEXOUT
DEC     BP
MOV     AL,[BP]             ;Low sec
CALL    AL_HEXOUT

```

```
        ;---- LBA FORMAT ----
```

```
MOV     BX,msgLBA
CALL    PRINT_STRING      ;(LBA = 00 (<-- Old "Heads" = 0 for these drives).

MOV     BP,RAM_DRIVE_TRK+1 ;High "cylinder" byte
MOV     AL,[BP]
CALL    AL_HEXOUT
DEC     BP
MOV     AL,[BP]            ;Low "cylinder" byte
CALL    AL_HEXOUT

MOV     BP,RAM_DRIVE_SEC
MOV     AL,[BP]
CALL    AL_HEXOUT
MOV     BX,MSGBracket      ;)$
CALL    PRINT_STRING
RET
```

```
SHOW_TRACK_SEC:                ;Display current (CPM) track,sector
```

```
MOV     BX,msgCPMTRK
CALL    PRINT_STRING      ;---- CPM FORMAT ----
MOV     BP,RAM_TRK+1
MOV     AL,[BP]            ;High TRK byte
CALL    AL_HEXOUT
DEC     BP
```



```
MOV     AL,[BP]                ;Low TRK byte
```

```
CALL    AL_HEXOUT
```

```
MOV     BX,msgCPMSEC
```

```
CALL    PRINT_STRING
```

```
MOV     BP,RAM_SEC            ;Low Sec (Only)
```

```
MOV     AX,[BP]
```

```
CALL    AL_HEXOUT
```

```
MOV     BX,H_Msg
```

```
CALL    PRINT_STRING
```

```
RET
```

```
SHOW_TRACK:
```

```
MOV     BX,msgCPMTRK
```

```
CALL    PRINT_STRING          ;---- CPM FORMAT ----
```

```
MOV     BP,RAM_TRK+1
```

```
MOV     AL,[BP]                ;High TRK byte
```

```
CALL    AL_HEXOUT
```

```
MOV     BP,RAM_TRK
```

```
MOV     AL,[BP]                ;Low TRK byte
```

```
CALL    AL_HEXOUT
```

```
MOV     BX,OK_CR_MSG
```

```
CALL    PRINT_STRING          ;---- CPM FORMAT ----
```

```
RET
```

```

DISPLAY_SEC:                ;Print a DISPLAY_SEC of the data in the 512 byte IDE_Buffer (RAM_DMA)

    CALL    CRLF            ;Note written so it can be easily converted to a "normal: DS: based" routine

    MOV     BP, RAM_DMA     ;Get Current DMA Address

    MOV     SI, [BP]        ;Both DS:DI & SI point to buffer

    MOV     DI, SI

    MOV     DH, 32          ;print 32 lines

SF172: CALL    CRLF

    call    SHOW_ADDRESS_SS    ;Show SS:SI

    mov     cx, 2            ;send 2 spaces

    call    TABS

    MOV     DL, 16           ;32 characters across

SF175: MOV     AL, [SS:SI]

    CALL    AL_HEXOUT        ;Display A on CRT/LCD

    MOV     AL, '~'

    CALL    CO

    INC     SI

    DEC     DL

    JNZ     SF175

    mov     cx, 3            ;first send 3 spaces

    call    TABS

    MOV     DL, 16           ;24 across again

```

```

Sloop2:      mov     al,[SS:DI]

             and     al,7fh

             cmp     al,' '           ;filter out control characters

             jnc     Sloop3

Sloop4:      mov     al,'.'

Sloop3:      cmp     al,'~'

             jnc     Sloop4

             mov     cl,al

             call    CO

             INC     DI

             DEC     DL

             JNZ     Sloop2

             DEC     DH

             JNZ     SF172           ;--DH has total byte count

             CALL    CRLF

             ret

```

;Point to next sector. Ret Z if all OK NZ if at end of disk

GET_NEXT_SECT:

```

MOV     BP,RAM_SEC           ;Get Current Sector

MOV     AX,[BP]

INC     AX

MOV     [BP],AX              ;0 to MAXSEC CPM Sectors

```

```

    CMP     AX,MAXSEC-1           ;Assumes < 255 sec /track
    JNZ     NEXT_SEC_DONE

    MOV     AX,0                 ;Back to CPM sector 0
    MOV     [BP],AX

    MOV     BP,RAM_TRK          ;Bump to next track
    MOV     AX,[BP]
    INC     AX
    CMP     AX,100H              ;Tracks 0-0FFH only
    JZ      AT_DISK_END
    MOV     [BP],AX

NEXT_SEC_DONE:
    CALL    WR_LBA               ;Update the LBC pointer
    XOR     AX,AX
    RET                               ;Ret z if all OK

AT_DISK_END:
    XOR     AX,AX
    DEC     AX
    RET

```

```

;Point to previous sector.  Ret Z if all OK

```

```

GET_PREV_SECT:

```

```

MOV     BP, RAM_SEC           ;Get Current Sector
MOV     AX, [BP]
CMP     AX, 0
JZ      PREVIOUS_TRACK
DEC     AX
MOV     [BP], AX              ;0 to MAXSEC CPM Sectors
JMP     PREVIOUS_SEC_DONE

```

PREVIOUS_TRACK:

```

MOV     AX, MAXSEC-1          ;Back to CPM last sector on previous track
MOV     [BP], AX

MOV     BP, RAM_TRK          ;Bump to next track
MOV     AX, [BP]
CMP     AX, 0                 ;If On track 0 already then problem
JNZ     AT_00
DEC     AX
MOV     [BP], AX

```

PREVIOUS_SEC_DONE:

```

CALL    WR_LBA                ;Update the LBC pointer
XOR     AX, AX                ;Return Z if all OK
RET

```

AT_00: MOV BX, ATHOME_MSG

```
CALL    PRINT_STRING
```

```
XOR     AX, AX
```

```

    DEC     ax                ;NZ if problem
    RET

;

SHOWerrors:
    CALL    CRLF
    MOV     DL,REGstatus      ;Get status in status register
    CALL    IDErd8D
    MOV     AL,DH
    AND     AL,1H
    JNZ     MoreError        ;Go to REGerr register for more info
                                ;All OK if 01000000

    PUSHF                    ;<<< Save for return below
    AND     AL,80H
    JZ      NOT7
    MOV     BX,DRIVE_BUSY    ;Drive Busy (bit 7) stuck high.   Status =
    CALL    PRINT_STRING
    JMP     DONEERR

NOT7:  AND     AL,40H
    JNZ     NOT6
    MOV     BX,DRIVE_NOT_READY ;Drive Not Ready (bit 6) stuck low.  Status =
    CALL    PRINT_STRING
    JMP     DONEERR

```

```

NOT6:  AND    AL,20H

        JNZ    NOT5

        MOV    BX,DRIVE_WR_FAULT    ;Drive write fault.    Status =

        CALL   PRINT_STRING

        JMP    DONEERR

NOT5:   MOV    BX,UNKNOWN_ERROR

        CALL   PRINT_STRING

        JMP    DONEERR

MoreError:                                ;Get here if bit 0 of the status register indicted a problem

        MOV    DL,REGerr                ;Get error code in REGerr

        CALL   IDerd8D

        MOV    AL,DH

        PUSHF                            ;<<<< Save flags for below

        AND    AL,10H

        JZ     NOTE4

        MOV    BX,SEC_NOT_FOUND

        CALL   PRINT_STRING

        JMP    DONEERR

NOTE4:  AND    AL,80H

        JZ     NOTE7

```

```
MOV     BX,BAD_BLOCK
CALL    PRINT_STRING
JMP     DONEERR
```

```
NOTE7:  AND     AL,40H
        JZ      NOTE6
MOV     BX,UNRECOVER_ERR
CALL    PRINT_STRING
JMP     DONEERR
```

```
NOTE6:  AND     AL,4H
        JZ      NOTE2
MOV     BX,INVALID_CMD
CALL    PRINT_STRING
JMP     DONEERR
```

```
NOTE2:  AND     AL,2H
        JZ      NOTE1
MOV     BX,TRK0_ERR
CALL    PRINT_STRING
JMP     DONEERR
```

```
NOTE1:  MOV     BX,UNKNOWN_ERROR1
        CALL    PRINT_STRING
        JMP     DONEERR
```



```

DONEERR:POPF                ;>>>> get back flags

    PUSH    AX

    CALL    AL_BINOUT        ;Show error bit pattern

    CALL    CRLF

    POP     AX

    XCHG    AL,AH

    OR      AL,AL            ;Set Z flag

    STC                      ;Set Carry flag

    RET

```

```

;=====
; IDE Drive BIOS Routines written in a format that can be used with CPM86 (Note MSDOS/DOS has its own
; modules see further below. However instead of using DS:[BX] (as we do in the CPM86 BIOS), throughout we
; will use SS:[BP] so the the buffers can reside at the top segment of available RAM.
; Normally this will be D000:E000H (below the ROM) but the monitor will not assume that there is a
; full 1MG address space available and may put them lower. See monitor initialization code at start.
;=====

```

```

IDEinit:                    ;Initilze the 8255 and drive then do a hard reset on the drive,

                            ;By default the drive will come up initilized in LBA mode.

    MOV     AL,READcfg8255    ;10010010b

    OUT     IDECtrlPort,AL    ;Config 8255 chip, READ mode

```

```

MOV     AL,IDErstline
OUT     IDEportC,AL           ;Hard reset the disk drive

MOV     CH,IDE_Reset_Delay   ;;Time delay for reset/initilization (~66 uS, with 8MHz 8086, 1 I/O wait state)
ResetDelay:
DEC     CH
JNZ     ResetDelay           ;Delay (IDE reset pulse width)
XOR     AL,AL
OUT     IDEportC,AL         ;No IDE control lines asserted

CALL    DELAY_SHORT         ;Allow time for CF/Drive to recover

MOV     DH,11100000b         ;Data for IDE SDH reg (512bytes, LBA mode,single drive,head 0000)
; MOV   DH,10100000b         ;For Trk,Sec,head (non LBA) use 10100000 (This is the mode we use for MSDOS)
;                                     ;Note. Cannot get LBA mode to work with an old Seagate Medalist 6531 drive.
;                                     ;have to use the non-LBA mode. (Common for old hard disks).

MOV     DL,REGshd            ;00001110, (0EH) for CS0,A2,A1,
CALL    IDEwr8D              ;Write byte to select the MASTER device

MOV     CH,03H              ;<<< May need to adjust delay time
WaitInit:
MOV     DL,REGstatus         ;Get status after initilization
CALL    IDErd8D              ;Check Status (info in [DH])

```

```

MOV     AL,DH
AND     AL,80H
JZ      DoneInit          ;Return if ready bit is zero
CALL    DELAY_LONG        ;Long delay, drive has to get up to speed
DEC     CH
JNZ     WaitInit
XOR     AL,AL
DEC     AL
RET      ;Return NZ. Well check for errors when we get back

DoneInit:
RET      ;Return Z indicating all is well

DELAY_LONG:                ;Long delay (Seconds) for hard disk to get up to speed
PUSH    CX
PUSH    DX
MOV     CX,0FFFFH
DELAY2:  MOV     DH,2        ;May need to adjust delay time to allow cold drive to
DELAY1:  DEC     DH          ;to speed
JNZ     DELAY1
DEC     CX
JNZ     DELAY2
POP     DX
POP     CX
RET

```

DELAY_SHORT:

MOV AX,8000H ;DELAY ~32 MS (DOES NOT SEEM TO BE CRITICAL)

DELAY3: DEC AX

JNZ DELAY3

RET

;Read a sector, specified by the 4 bytes in LBA

;Z on success, NZ call error routine if problem

READSECTOR:

CALL WR_LBA ;Tell which sector we want to read from.

;Note: Translate first in case of an error otherwise we

;will get stuck on bad sector

CALL IDEwaitnotbusy ;make sure drive is ready

JNB L_19

JMP SHOWerrors ;Returned with NZ set if error

L_19: MOV DH,COMMANDread

MOV DL,REGcommand

CALL IDEwr8D ;Send sec read command to drive.

CALL IDEwaitdrq ;wait until it's got the data

JNB L_20

JMP SHOWerrors

```

L_20:  MOV     BP, RAM_DMA           ;Get Current DMA Address at SS:RAM_DMA
        MOV     AX, [BP]             ;Note SS: is assumed here
        MOV     BP, AX
        MOV     CH, 0               ;Read 512 bytes to [HL] (256X2 bytes)

```

```

MoreRD16:

```

```

        MOV     AL, REGdata          ;REG regsite address
        OUT     IDEportC, AL

        OR      AL, IDErdline        ;08H+40H, Pulse RD line
        OUT     IDEportC, AL

        IN      AL, IDEportA         ;Read the lower byte first
        MOV     [BP], AL
        INC     BP
        IN      AL, IDEportB         ;THEN read the upper byte
        MOV     [BP], AL
        INC     BP

        MOV     AL, REGdata          ;Deassert RD line
        OUT     IDEportC, AL
        DEC     CH
        JNZ     MoreRD16

```

```

MOV     DL,REGstatus
CALL    IDErd8D
MOV     AL,DH
AND     AL,1H
JZ      L_21
CALL    SHOWerrors           ;If error display status
L_21:   RET

;Write a sector, specified by the 3 bytes in LBA (_ IX+0)",
;Z on success, NZ to error routine if problem

WRITESECTOR:
CALL    WR_LBA               ;Tell which sector we want to read from.
;Note: Translate first in case of an error otherwise we
;will get stuck on bad sector

CALL    IDEwaitnotbusy       ;make sure drive is ready
JNB     L_22
JMP     SHOWerrors

L_22:   MOV     DH,COMMANDwrite
MOV     DL,REGcommand
CALL    IDEwr8D              ;tell drive to write a sector
CALL    IDEwaitdrq           ;wait unit it wants the data
JNB     L_23
JMP     SHOWerrors

```

```

L_23:  MOV     BP, RAM_DMA           ;Get Current DMA Address
      MOV     AX, [BP]
      MOV     BP, AX
      MOV     CH, 0                 ;256X2 bytes

      MOV     AL, WRITEcfg8255
      OUT     IDECtrlPort, AL

WRSEC1_IDE:
      MOV     AL, [BP]
      INC     BP
      OUT     IDEportA, AL          ;Write the lower byte first
      MOV     AL, [BP]
      INC     BP
      OUT     IDEportB, AL          ;THEN High byte on B

      MOV     AL, REGdata
      PUSH    AX
      OUT     IDEportC, AL          ;Send write command
      OR      AL, IDEwrlne          ;Send WR pulse
      OUT     IDEportC, AL
      POP     AX
      OUT     IDEportC, AL          ;Send write command
      DEC     CH

```

```

JNZ    WRSEC1_IDE

MOV     AL,READcfg8255           ;Set 8255 back to read mode
OUT     IDECtrlPort,AL

MOV     DL,REGstatus
CALL    IDErd8D
MOV     AL,DH
AND     AL,1H
JZ      L_24
CALL    SHOWerrors              ;If error display status

L_24:  RET

WR_LBA:                                ;Write the logical block address to the drive's registers
                                           ;Note we do not need to set the upper nibble of the LBA
                                           ;It will always be 0 for these small CPM drives (so no High Cylinder
                                           ;numbers etc).

MOV     BP,RAM_SEC
MOV     AX,[BP]                  ;LBA mode, Low sectors go directly
INC     AX                      ;Sectors are numbered 1 -- MAXSEC (even in LBA mode)
MOV     BP,RAM_DRIVE_SEC
MOV     [BP],AL                 ;For Diagnostic Diaplay Only
MOV     DH,AL
MOV     DL,REGsector            ;Send info to drive

```



```

CALL    IDEwr8D                ;Write to 8255 A Register

                                ;Note: For drive we will have 0 - MAXSEC sectors only

MOV     BP, RAM_TRK
MOV     AX, [BP]
MOV     BP, RAM_DRIVE_TRK
MOV     [BP], AL
MOV     DH, AL                ;Send Low TRK#
MOV     DL, REGcylinderLSB
CALL    IDEwr8D                ;Write to 8255 A Register

MOV     BP, RAM_DRIVE_TRK+1
MOV     [BP], AH
MOV     DH, AH                ;Send High TRK#
MOV     DL, REGcylinderMSB
CALL    IDEwr8D                ;Send High TRK# (in DH) to IDE Drive
CALL    IDEwr8D_X              ;Special write to 8255 B Register (Not A) to update LED HEX Display
                                ;High 8 bits ignored by IDE drive

MOV     DH, 1                 ;For CPM, one sector at a time
MOV     DL, REGsecnt
CALL    IDEwr8D                ;Write to 8255 A Register
RET

                                ;Special version for MS-DOS system BIOS (see IBM BIOS Section)

```

```

DOS_WR_LBA:                                ;This will display Head, Cylinder and Sector on the LED HEX display
                                           ;instead of LBA sector numbers.

      MOV    DH,[CURRENT_HEAD]              ;OR in head info to lower 4 bits
      AND    DH,0FH                        ;Just in case
      OR     DH,10100000B                   ;Set to >>>> NON-LBA mode <<<<
      MOV    DL,REGshd                      ;Send "Head #" (in DH) to IDE drive
      CALL   IDEwr8D

      MOV    DH,[CURRENT_TRACK_HIGH]        ;Send High TRK#
      MOV    DL,REGcylinderMSB
      CALL   IDEwr8D                        ;Send High TRK# (in DH) to IDE Drive

      MOV    DH,[CURRENT_HEAD]              ;Get head info to lower 8 bits of the special
      AND    DH,0FH                        ;top two LED HEX displays.
      SHL    DH,1                          ;These 8 (high) data lines are ignored by the IDE drive
      SHL    DH,1
      SHL    DH,1
      SHL    DH,1
      OR     DH,[CURRENT_TRACK_HIGH]        ;Will display the Head in top nibble and the two bits of the HIGH bits
      MOV    DL,REGcylinderMSB              ;of the high cylinder in the low nibble.
      CALL   IDEwr8D_X                      ;Special output to 8255 B Register (Not A) to update LED HEX Display ONLY

      MOV    DH,[CURRENT_TRACK]             ;Get low Track #
      MOV    DL,REGcylinderLSB              ;Send Low TRK# (in DH)
      CALL   IDEwr8D                        ;Special write to 8255 B Register (Not A)

```

```

MOV    DH,[CURRENT_SECTOR]      ;Bits 0-5 only  (currently 1-17)
MOV    DL,REGsector              ;Send "Sector#"
CALL   IDEwr8D                   ;Write to 8255 A Register

```

```

MOV    DH,[SECTORS_TO_DO]        ;# of CONTIGOUS sectors to send
MOV    DL,REGsecCnt              ;
CALL   IDEwr8D                   ;Write to 8255 A Register
RET

```

```

IDEwaitnotbusy:                  ;Drive READY if 01000000

```

```

MOV    CH,0FFH
MOV    AH,0FFH                  ;Delay, must be above 80H for 4MHz Z80. Leave longer for slower drives
PUSH   BX                       ;AH is not changed in IDErd8D below

```

```

MoreWait:

```

```

MOV    DL,REGstatus             ;wait for RDY bit to be set
CALL   IDErd8D                  ;Note AH or CH are unchanged
MOV    AL,DH
AND    AL,11000000B
XOR    AL,01000000B
JZ     DONE_NOT_BUSY
DEC    CH
JNZ    MoreWait

```

```

DEC    AH
JNZ    MoreWait
STC                                ;Set carry to indicate an error
POP    BX
RET

```

DONE_NOT_BUSY:

```

OR     AL,AL                      ;Clear carry it indicate no error
POP    BX
RET

```

;Wait for the drive to be ready to transfer data.

IDEwaitdrq: ;Returns the drive's status in Acc

```

MOV    CH,0FFH
MOV    AL,0FFH                   ;Delay, must be above 80H for 4MHz Z80. Leave longer for slower drives
PUSH   BX

```

MoreDRQ:

```

MOV    DL,REGstatus              ;wait for DRQ bit to be set
CALL   IDErd8D                   ;Note AH or CH are unchanged
MOV    AL,DH
AND    AL,10001000B
CMP    AL,00001000B
JZ     DoneDRQ
DEC    CH
JNZ    MoreDRQ

```

```

    DEC     AH
    JNZ     MoreDRQ
    STC                                ;Set carry to indicate error
    POP     BX
    RET

DoneDRQ:
    OR      AL,AL                      ;Clear carry
    POP     BX
    RET

CLEAR$ID$BUFFER:                      ;Clear the ID Buffer area
    MOV     AX,2020H                  ;Clear to spaces
    MOV     BP,IDE_Buffer             ;Remember CS: = SS
    MOV     CX,256                    ;512 bytes total

CLEAR2:    MOV     [BP],AX             ;Note this will be SS:[BP]
    INC     BP
    INC     BP
    LOOP    CLEAR2

    MOV     AX,0H                     ;Put in 0's for cylinder,heads,sectors etc
    MOV     BP,IDE_Buffer
    MOV     CX,7                      ;14 bytes total

CLEAR3:    MOV     [BP],AX             ;Note this will be SS:[BP]
    INC     BP

```

```
INC     BP
LOOP    CLEAR3
RET
```

```

;-----
; Low Level 8 bit R/W to the drive controller.  These are the routines that talk
; directly to the drive controller registers, via the 8255 chip.
; Note the 16 bit Sector I/O to the drive is done directly
; in the routines READSECTOR & WRITESECTOR for speed reasons.

```

```

IDEr8D:                                ;READ 8 bits from IDE register @ [DL], return info in [DH]

    MOV     AL,DL                       ;select IDE register

    OUT     IDEportC,AL                 ;drive address onto control lines


    OR      AL,IDERdline               ;RD pulse pin (40H)

    OUT     IDEportC,AL                 ;Assert read pin


    IN      AL,IDEportA

    MOV     DH,AL                      ;return with data in [DH]


    MOV     AL,DL                      ;<---Ken Robbins suggestion

    OUT     IDEportC,AL                 ;Drive address onto control lines

```

```

XOR    AL,AL
OUT     IDEportC,AL      ;Zero all port C lines
RET

```

```

IDEWr8D:                                ;WRITE Data in [DH] to IDE register @ [DL]

MOV     AL,WRITEcfg8255                ;Set 8255 to write mode
OUT     IDECtrlPort,AL

MOV     AL,DH                          ;Get data put it in 8255 A port
OUT     IDEportA,AL

MOV     AL,DL                          ;select IDE register
OUT     IDEportC,AL

OR      AL,IDEwrlne                    ;lower WR line
OUT     IDEportC,AL

MOV     AL,DL                          ;<-- Ken Robbins suggestion, raise WR line
OUT     IDEportC,AL                ;deassert RD pin

XOR     AL,AL                          ;Deselect all lines including WR line
OUT     IDEportC,AL

MOV     AL,READcfg8255                 ;Config 8255 chip, read mode on return

```

```
OUT    IDECtrlPort,AL
```

```
RET
```

```
IDEWr8D_X:                ;WRITE Data in [DH] to IDE register @ [DL]
```

```
MOV    AL,WRITEcfg8255      ;Set 8255 to write mode
```

```
OUT    IDECtrlPort,AL
```

```
MOV    AL,DH                ;Get data and put it in 8255 >>>> Port B <<<<
```

```
OUT    IDEportB,AL
```

```
MOV    AL,DL                ;select IDE register
```

```
OUT    IDEportC,AL
```

```
OR     AL,IDEwrline         ;lower WR line
```

```
OUT    IDEportC,AL
```

```
MOV    AL,DL                ;<-- Ken Robbins suggestion, raise WR line
```

```
OUT    IDEportC,AL          ;Deassert RD pin
```

```
XOR    AL,AL                ;Deselect all lines including WR line
```

```
OUT    IDEportC,AL
```

```
MOV    AL,READcfg8255        ;Config 8255 chip, read mode on return
```

```
OUT    IDECtrlPort,AL
```


RET

```
;*****
;
;   "BIOS" section to allow MS-DOS 4.1 to run on non-IBM hardware.
;   8086 assembly language for the NASM assembler. This is a highly
;   modified version of a BIOS first written by LogiCom Inc back in 1985.
;
;*****
;
;   The normal interrupts for the IBM, and their entry points
;   in this code are as follows:
;
;   Int   Name                BIOS entry
;   ---   ---                ---
;   0     Divide by zero      DUMMY_RETURN
;   1     Single Step         DUMMY_RETURN
;   2     Non-maskable        NMIINT
;   3     Breakpoint          DUMMY_RETURN
;   4     Overflow            DUMMY_RETURN
;   5     Print Screen        DUMMY_RETURN
;   6     Reserved            DUMMY_RETURN
```

```

; 7  Reserved          DUMMY_RETURN
; 8  Timer Tic         TIMER          \
; 9  Keypressed        KEYHND         \
; A  Reserved          DUMMY_RETURN   \
; B  Comm Hardware     DUMMY_RETURN   \ Normal location for
; C  Comm Hardware     DUMMY_RETURN   / IBM hardware interrupts
; D  Disk Hardware     DUMMY_RETURN   /
; E  Diskette Hardware DUMMY_RETURN   /
; F  Printer Hardware  DUMMY_RETURN   /

; 10 Video Output      CONOUT          (10 through 1F are
; 11 Equipment check   EQUIP           software interrupts)
; 12 Memory Size       MEMSIZ
; 13 Disk I/O          DISKIO          <----- ALL DISK IO (Floppy & HDISK)
; 14 Comm I/O          COMMIO
; 15 Cassette I/O      DUMMY_RETURN
; 16 Keyboard I/O      CONIN
; 17 Printer I/O       LSTOUT
; 18 Basic             DUMMY_RETURN
; 19 Bootstrap         BOOT_DOS_INT
; 1A Time of Day       TIME_OF_DAY
; 1B Keyboard Break    DUMMY_RETURN
; 1C User timer tic    DUMMY_RETURN
; 1D Video Init.       VIDEO_PARM
; 1E Diskette Parms    DISK_BASE (Pointer only)

```

```
; 1F  Graphics Char      0
;
; 40  Copy of Disk/IO    DISKIO (for systems with a HDISK)
```

```
IBM_BIOS:
```

```
cli                      ;No interrupts yet please

MOV  BX,IBM_SIGNON_MSG   ;Announce we are here

CALL PRINT_STRING        ;Note PRINT_STRING always uses the CS: override for the BX pointer

push  DS

XOR  AX,AX                ;Set DS to data area for ROM usage in low RAM @ 400H....)

MOV  DS,AX

mov  byte [DEBUG_FLAG],0 ;Debug mode normally off

POP  DS

CALL SETUP_IBM_BIOS      ;Initilize RAM and hardware
```

```
IBM_LOOP:
```

```
CALL  CRLF

MOV  BX,IBM_MENU1        ;Enter start of menu

CALL  PRINT_STRING

XOR  AX,AX                ;Set DS to data area for ROM usage in low RAM @ 400H....)

MOV  DS,AX
```

```

    CMP     byte [DEBUG_FLAG],0 ;Debug mode (normally off)
    POP     DS

    JNZ     MENU_ON

    MOV     BX,IBM_MENU_OFF      ;Enter "OFF"
    CALL    PRINT_STRING
    JMP     IBM_LOOP1

MENU_ON:
    MOV     BX,IBM_MENU_ON      ;Enter "ON"
    CALL    PRINT_STRING

IBM_LOOP1:
    MOV     BX,IBM_MENU2      ;Enter the rest of the menu
    CALL    PRINT_STRING

    MOV     AX,CS
    MOV     DS,AX             ;Just to be safe for below

    call    CICO              ;Get a command from Console
    mov     ah,0

    CMP     AL,ESC            ;Abort if ESC
    JNZ     NOT_ESC_IBM

    JMP     INIT              ;Back to start of Monitor

NOT_ESC_IBM:
    cmp     al,'A'            ;Find menu option from table
    jb      IBM_LOOP          ;must be A to Z

```

```

cmp    al,'Z'
jg     IBM_LOOP
sub    al,'A'           ;calculate offset
shl    al,1             ;X 2
add    ax,IBM_TABLE     ;Note DS:=CS: in this monitor by default
mov    bx,ax
CALL   CRLF
mov    ax,[cs:bx]       ;get location of routine CS:[BX]
call   ax               ;<----- This is the PC-BIOS Menu CMD call
jmp    IBM_LOOP         ;finished

```

```

;----- Initilize RAM and hardware to look like an IBM-PC setup

```

```

;XXXX:

```

```

;

```

```

SETUP_IBM_BIOS:

```

```

    mov    ax,cs

```

```

    mov    ds,ax         ;DS is this ROM's CS

```

```

    sub    ax,ax

```

```

    mov    es,ax         ;ES: = 0H in RAM for STOW's below

```

```

    CLD                ;Default to direction up

```

```

    CALL   SETUP_INT_TABLE      ;Fill all RM 8086 interrupts initially with a default error trapping pointer

```

```

mov     di,PrintScreen           ;Setup PrintScreen vector in low RAM (at 14H)
mov     ax,PrintScreenRoutine    ;Have it point to the relevent return in this monitor
stosw                                   ;(ES: used for final location)

mov     cx,8                      ;Set all 8 hardware interrupts for 8259A (at I/O port address 20H)
mov     si,vec_tbl_8259A ;Move the pointers in vec_tbl-8259A to low RAM starting at 20H
mov     di,Start8259A_Ints       ;Note DS:(=CS:) is source,   ES: is destination
iloop1: movsw
inc     di                        ;Skip over the segment pointer (already done above), to next vector offset
inc     di
loop    iloop1

mov     cx,16                    ;Set all 16 software interrupts
mov     si,vec_tbl_soft_ints
mov     di,CRTINT                ;Start location in low RAM
iloop2: movsw                    ;Note DS: (=CS:) is source,   ES: is destination
inc     di                        ;Skip over the segment pointer (already done above), to next vector offset
inc     di
loop    iloop2

MOV     CX,IVGA_VAL_LEN          ;Some RAM variables need to be initilized for S-100 Lomas CGA Board (@0:449H)
MOV     SI,INITIAL_VGA_VALUES
MOV     DI,CRT_MODE              ;0:449H
iloop3: movsw                    ;Note DS: (=CS:) is source,   ES: is destination

```

```

loop    iloop3                ;To be safe, do not use "rep" in case marginal EPROM's cannot handle speed.

IN      AL,IOBYTE

TEST    AL,2                  ;Bit 1 of IOBYTE Port will force output to CGA/VGA

JNZ     NO_FORCE_CGA_DISPLAY  ;is there a request to switch CRT outputs

MOV     word [ES:CONSOL_FLAG],1 ;1= Force console output to CGA/VGA Video Board upon MS_DOS Bootup,
TO_VGA: MOV     BX,VIDIO_VGA_MSG ;Video to VGA

CALL    PRINT_STRING

JMP     DONE_SWAP

NO_FORCE_CGA_DISPLAY:

CMP     word [ES:CONSOL_FLAG],1 ;Already requested by "B" menu main command?

JZ      TO_VGA

CMP     word [ES:CONSOL_FLAG],2 ;Is output already set for LAVA board by "B" menu main command?

JNZ     TO_PROP_BRD          ;Not 2, then definately not LAVA

MOV     BX,VIDIO_LAVA_MSG     ;Announce we are going to Lava Board. (There is a 1/32K chance RAM had 02H at CONSOL_FLAG at
startup)

CALL    PRINT_STRING          ;Note PRINT_STRING always uses the CS: override for the BX pointer

JMP     DONE_SWAP

TO_PROP_BRD:

MOV     word [ES:CONSOL_FLAG],0 ;Default, Force Console output to Propeller Video Board if anything else

```

```

DONE_SWAP:                ;The 496H-490H area needs to be 0's for MS-DOS 4.01

                            ;Clear the whole IBM-AT "extra store area" to 0's

                            ;It seems MSDOS V4.01 counts on at least the diskette area being 0's

    mov     cx,(496H-490H)    ;It hangs on a boot otherwise!  Count of bytes in the area

    mov     di,DSK_STATE

    xor     AX,AX

iloop4: MOV     [ES:DI],AX    ;ES: = 0, is destination

    inc     di                ;Skip over the segment pointer (already done above), to next vector offset

    inc     di

    loop    iloop4

                            ;Now a few special case situations...

                            ;Note, in every case ES:=0, is destination segment


    MOV     AX,CRT_CHAR_GEN    ;7CH, Upper 128 Bytes of 256 Byte character set

    mov     DI,EXT_CHAR_PTR    ;(1FH*4) (Note no valid table is actually present for 8088/8086 EEPROMS)

    stosw


    MOV     AX,OLD_DISKIO      ;100H, We need to handle software Int 40H (The relocated old INT 13H PC Bios Floppy I/O)

    mov     DI,OLD_DISK_VEC    ; (40H*4)

    stosw


    MOV     AX,FDISK_3PARM_TBL ;We need to move the boot diskette parameter table to Int 1EH*4 area. (Use 1.44M 3" Floppy)

    mov     DI,FDISK_PARMS

    stosw

```



```

MOV  AX,HDISK_PARM_TBL    ;104H, Setup the default HARD DISK #1, table POINTER offset
mov  DI,HDISK_PARMS       ; (41H*4) ,
stosw

```

```

MOV  AX,CRT_CHAR_GEN      ;10CH, 256 Byte character set
mov  DI,EXT_CHAR_PTR2     ; (43H*4)
stosw

```

```

MOV  AX,HDISK_PARM_TBL    ;118H, Setup the same default HARD DISK #2, table POINTER offset
mov  DI,HDISK2_PARMS      ; (46H*4)
stosw                     ;ES: 0, is destination segment

```

```

;Now set up the memory variables

```

```

XOR   AX,AX               ;Now set DS: (=0) to data area for ROM usage in low RAM @400H
MOV   DS,AX
mov   word [expram],msize-64 ;show expansion ram size
mov   word [memrsz],msize   ;and total memory size (640K)
mov   word [EQFLAG],0100001001100001B ;set equipment flag so IBM is happy

```

```

;bit 0    disk drives present
;bit 1    8087 Present
;bit 2    Mouse present

```

```

;bit 3      ----
;bits 4,5   default to colour card
;           00 EGA
;           01 40X25 Color
;           10 80X25 Color
;           11 80X25 Monochrome
;bits 6,7   floppy drives -1 (if bit 0 =1)
;bit 8      DMA support installed (PCjr, Tandy)
;bits 9,10,11 number of serial ports
;bit 12     no game adaptor
;bit 13     serial printer attachd (PCjr)
;bits 14,15 no of printers

mov     ax,keybuff           ;keyboard interrupt pointers
mov     [bufhd],ax
mov     [buftl],ax
mov     byte [chrcnt],0

mov     byte [VERIFY_FLAG],0 ;Initially set for sector reads (rather than sector verifies)

;Initilize hardware to emmulate IBM-PC settings

mov     bx,PIC_INIT_MSG      ;Send a signon about initilizing the 8259A
call    PRINT_STRING

mov     al,MasterICW1        ;Initilize the 8259A PIC Controller

```

```

out    MASTER_PIC_PORT,al
mov    al,MasterICW2        ;Ints starts at 20H in RAM
out    MASTER_PIC_PORT+1,al
mov    al,MasterICW4        ;No slaves above, so 8259 does not expect ICW3
out    MASTER_PIC_PORT+1,al

mov    al,11111111b         ;No V0 & V1 for now
out    MASTER_PIC_PORT+1,al

                                ;Initilize the timer
MOV    AL,36H                ;Sel TIM 0, LSB,MSB,Mode 3
OUT    TIM_CTL,AL            ;Write to Timer mode register (43H)
MOV    AL,0
OUT    TIMER,AL              ;LSB = 0
OUT    TIMER,AL              ;MSB = 0

                                ;Next move the current time into the system tick bytes in low RAM
                                ;Remember DS: is already set to data area for ROM usage in low RAM (400H)

mov    word [timlow],0        ;Default setup timer/RTC default values
mov    word [timhi],0
mov    word [timofl],0        ;Set clock tick info to 0 in low ram

MOV    AL,CMOS_VALID         ;Before getting current CMOS time check chip is there and working
OUT    CMOS_PORT,AL

```

```

    JMP     SHORT $+2          ;Delay

    IN      AL,CMOS_PORT+1

    SUB     AL,80H              ;Check bad battery is OK. (Note different from AT. Dallas DS12887 says valid = 80H)

    JNZ     TOD_ERROR          ;If Not valid leave timer at 0

                                ;
                                ;
    SUB     CX,CX

UIP:  MOV     AL,CMOS_REGA

    OUT     CMOS_PORT,AL

    JMP     SHORT $+2

    IN      AL,CMOS_PORT+1

    TEST    AL,UPDATE_TIMER

    JZ      READ_SECONDS

    LOOP    UIP

    JMP     TOD_ERROR1         ;CMOS clock "stuck"

READ_SECONDS:

    MOV     AL,CMOS_SECONDS

    OUT     CMOS_PORT,AL

    JMP     SHORT $+2

    IN      AL,CMOS_PORT+1

                                ;
                                ;
    CMP     AL,059H            ;within range 0-59

    JA      TOD_ERROR2

                                ;
                                ;
    CALL    CVT_BINARY

    MOV     BL,COUNTS_SEC

```

```

MUL     BL
MOV     CX,AX
MOV     AL,CMOS_MINUTES
OUT     CMOS_PORT,AL
JMP     SHORT $+2
IN      AL,CMOS_PORT+1

CMP     AL,059H           ;Within range 0-59
JA      TOD_ERROR2

CALL    CVT_BINARY
MOV     BX,COUNTS_MIN     ;1092
MUL     BX
ADD     AX,CX
MOV     CX,AX
MOV     AL,CMOS_HOURS
OUT     CMOS_PORT,AL
JMP     SHORT $+2
IN      AL,CMOS_PORT+1
CMP     AL,023H           ;0-23
JA      TOD_ERROR2
CALL    CVT_BINARY
MOV     DX,AX
MOV     BL,COUNTS_HOUR    ;7
MUL     BL

```

```

        ADD     AX,CX

        ADC     DX,0000H

mov     [timhi],DX        ;Store in RAM
mov     [timlow],AX

;   mov     DX,0080H
;   mov     AX,340H
;   mov     [timhi],DX        ;Store in RAM
;   mov     [timlow],AX

        JMP     TOD_DONE

TOD_ERROR:
        MOV     BX,CMOS_CLOCK_MSG    ;Error reading CMOS Clock chip
        CALL    PRINT_STRING
        JMP     TOD_DONE

TOD_ERROR1:
        MOV     BX,CMOS_STUCK_MSG    ;Error reading CMOS Clock chip
        CALL    PRINT_STRING
        JMP     TOD_DONE

TOD_ERROR2:
        MOV     BX,CMOS_RANGE_MSG    ;Error reading CMOS Clock chip
        CALL    PRINT_STRING
        JMP     TOD_DONE

```

TOD_DONE:

```

    CLI

    IN      AL,MASTER_PIC_PORT+1    ;Allow timer tick

    AND     AL,0FEH

    OUT     MASTER_PIC_PORT+1,AL

    STI

```

INIT_VGA: ;We will try and initilize the CGA/VGA video board RAM area/ports anyway (even if not used)

```

    MOV     AX,0B800H                ;Segment of CGA Board RAM

    mov     di,[EQFLAG]

    and     di,30h                    ;Isolate crt switches (This is what IBM PC has - not used here!)

                                        ;bits 4,5  default to colour card
                                        ;
                                        ;          00 EGA
                                        ;
                                        ;          01 40X25 Color
                                        ;
                                        ;          10 80X25 Color
                                        ;
                                        ;          11 80X25 Monochrome

    cmp     di,30h

    jne     INIT_VGA1

    mov     ax,0B000h                ;Segment for Monochrome card

```

INIT_VGA1:

```

    mov     es,ax                    ;Set ES: to point to video area

    MOV     AL,03H                    ;Default to 80X25 Color

    MOV     DI,0                      ;CGA/VGA Board. If B/W card then DI = 30H

```

```

CALL    VGA_INIT                ;<<<<<<  Initilize the video board to 80X25 >>>>>>

                                ;This seem OK because IBM-CGA board comes up fine with the
                                ;ISA->S100 Adaptor board. Also the Lomas S-100 CGA board comes up fine.

MOV     AH,0H                   ;Initilize Serial Port (Used for debugging display if required)
MOV     AL,80H                  ;This sets for 9600 Baud. (However we will run at 38,400, see INT 14H)
MOV     DX,0

    int      14H                ;Serial out Handler    (Software Interrupt 14H)

IN      AL,IOBYTE               ;Allow IOBYTE to abort initilizing extra ROMS starting at C0000H
TEST    AL,4

    JNZ     VGA_ROM_CHECK       ;<<<< Check for VGA ROM at C000:0000H >>>>
RET                                           ;Else skip/ignore looking for a VGA ROM

                                ;Next, check if the is an extra ROM's/Software on board. This follows the IBM
                                ;format by looking at C0000H-F6000H (on 2K pages) for 55H,AAH and (length/512)
                                ;in the 3rd byte. (Note this BIOS is larger, should stop at D0000H)
                                ;We will just force initilization of the one VGA ROM if present, at C0000H
                                ;Can add more if required later.

                                ;If a valid "ROM" then we do a Call Far to byte 3 in that ROM/Code.

VGA_ROM_CHECK:                    ;It assumes that the code there will finish with a far return.

MOV     DX,C0000H                ;Will just look for Video ROM, so start at C0000H
MOV     DS,DX                    ;DS=C000H

```



```

SUB    BX,BX

MOV    AX,[BX]                ;[DS:BX] C000:0H

CMP    AX,0AA55H              ;Is the indicator flag there

JNZ    NO_ROM

;-----

MOV    BX,ROMCHECK_MSG        ;Announce we found a ROM at C000:0H

CALL  PRINT_STRING            ;Remember PRINT_STRING always uses the CS: override for the BX pointer
                                ;so no need to mess with DS

;-----

MOV    AX,40H                 ;Set ES=DSEG

MOV    ES,AX                  ;ES=40H

PUSH   DX                     ;Set things up just like IBM did (Just in case)!

MOV    word [ES:IO_ROM_INIT],0003H ;Offset @ 467H

MOV    [ES:IO_ROM_SEG],DS      ;Load segment @ 469H

;-----

CALL  0C000H:0003H            ;<<< Initilize EGA/VGA ROM >>>>
                                ;Note we assume there are no other extra ROM's

;-----

PUSH   AX                     ;Just in case values are needed

PUSH   BX

MOV    BX,ROMCHECK_MSG_OK      ;VGA ROM Initilized, returned back to BIOS.

CALL  PRINT_STRING            ;Note PRINT_STRING always uses the CS: override for the BX pointer

POP    BX

POP    AX

```

```

    POP     DX                      ;From above

    mov     ax,40h                  ;Test for EGA/VGA
    mov     es,ax
    mov     ah,12h
    mov     bx,0FF10h
    int     10h                    ;Video Get EGA Info
    cmp     BH,0FFh                ;If EGA or later present BH != FFh
    je      not_ega
    and     byte[es:EQFLAG],11001111b ;Set video flag in equipment list to EGA/VGA
                                           ;bits 4,5  default to colour card
                                           ;          00 EGA
                                           ;          01 40X25 Color
                                           ;          10 80X25 Color
                                           ;          11 80X25 Monochrome

not_ega:
    mov     ah,1
    mov     ch,0F0h
    int     10h                    ;Set cursor type
    call    clear_screen           ;clear display
    push    cs
    pop     ds
    RET

```

```

NO_ROM:

```

```

MOV     BX,NO_VGA_MSG           ;Announce no VGA ROM present

CALL PRINT_STRING               ;Note PRINT_STRING always uses the CS: override for the BX pointer

RET

```

clear_screen:

```

mov     dx,184Fh                ; Lower right corner of scroll
xor     cx,cx                   ; Upper left corner of scroll

mov     ax,600h                 ; Blank entire window

mov     bh,7                    ; Set regular cursor

int     10h                     ; Call video service scroll

mov     ah,2                    ; Set cursor position

xor     dx,dx                   ; upper left corner

mov     bh,0                    ; page 0

int     10h                     ; call video service

mov     ax,0500h                ; Set active display page zero

int     10h

ret

```

CVT_BINARY: ;Convert BCD in [AL] to Binary in [AL]

```

PUSH    CX

PUSH    AX

AND     AX,0FH

MOV     CX,AX                   ;Save low digit

POP     AX

```

```
PUSH    CX                ;On Stack
```

```
MOV     CL,4
```

```
ROR     AX,CL
```

```
AND     AL,0FH
```

```
MOV     CL,10
```

```
MUL     CL
```

```
POP     CX
```

```
ADD     AX,CX            ;Add in low digit
```

```
POP     CX
```

```
RET
```

```
;----- Menu CMD to Boot MS-DOS from a Floppy Disk using this BIOS
```

```
MMENU_FBOOT_DOS:        ;Come here from main menu. Debug mode ALWAYS off
```

```
push    DS
```

```
PUSH    AX
```

```
XOR     AX,AX            ;Set DS to data area for ROM usage in low RAM @ 400H....)
```

```
MOV     DS,AX
```

```
mov     [DEBUG_FLAG],AX  ;Debug mode normally off (AX=0)
```

```
POP     AX
```

```
POP     DS
```

```

MENU_FBOOT_DOS:                ;Come here from IBM BIOS menu (Debug mode MAY be on)

    MOV     BX,FBOOT_DOS_MSG    ;Booting MS-DOS

    CALL    PRINT_STRING

    MOV     DL,0                ;Make sure bit 7 is 0 for Floppy

    PUSH    DX                  ;Save value in DX (DL=0 for Floppy Boot)

```

```

COMMON_BOOT_DOS:                ;Common BOOT MS-DOS/FreeDOS entry point

    CALL    SETUP_IBM_BIOS      ;Initilize RAM and hardware

    mov     al,11111100b        ;Allow S-100 bus ints V0 & V1 (only) now

    out     MASTER_PIC_PORT+1,al

    sti                                ;Enable hardware interrupts

    POP     DX                  ;Get back Floppy/HDISK info

    int     19H                 ;<<<<<<< Boot PC-DOS with software int 19H

                                ;Should never return here IF no problem

    JMP     word 0F000H:INIT     ;Far Jump to F000H:INIT (Start of this monitor)

```

```

;----- Menu CMD to Boot MS-DOS from a HARD Disk using this BIOS

```

MMENU_HBOOT_DOS:

```

    push    DS
    PUSH    AX
    XOR     AX,AX                ;Set DS to data area for ROM usage in low RAM @ 400H....)
    MOV     DS,AX
    mov     [DEBUG_FLAG],AX      ;Debug mode normally off (AX=0)
    POP     AX
    POP     DS

```

MENU_HBOOT_DOS:

```

    MOV     BX,HBOOT_DOS_MSG     ;Booting MS-DOS
    CALL    PRINT_STRING
    MOV     DX,0080H             ;Make sure bit 7 is 1 for HDisk
    PUSH    DX                   ;Save value in DX (DL=80H for HDisk Boot)
    JMP     COMMON_BOOT_DOS

```

;----- Menu CMD to test 8259A Interrupt driven Keypress code using this IBM BIOS

MENU_KEY_TEST:

```

    MOV     BX,KEY_TEST_MSG      ;Keyboard test
    CALL    PRINT_STRING

    mov     al,11111101b         ;Allow V1 on 8259A now

```

```

out    MASTER_PIC_PORT+1,al

sti                                ;Enable hardware interrupts

Next_Key:
    MOV    BX,IN_CHAR_MSG          ;Input character =
    CALL   PRINT_STRING

Next_Key1:
    MOV    AH,01H                  ;Check if anything there
    int     16H                    ;Get Keyboard status. Console Input Handler    (Software Interrupt 16H)
    JZ     Next_Key1

    MOV    AH,0H                    ;Get actual character from buffer
    int     16H                    ;Get Character. Console Input Handler    (Software Interrupt 16H)

    CMP    AL,ESC
    JZ     Key_Done
;    MOV    CL,AL
;    CALL   CO                      ;Send direct to Console
    CALL   AL_HEXOUT                ;Display the hex character recieved
    MOV    BX,GOT_CHAR_MSG          ;Recieved character =
    CALL   PRINT_STRING
    JMP    Next_Key

Key_Done:

```

```

mov     al,11111111b           ;Do not Allow V1 on 8259A again
out     MASTER_PIC_PORT+1,al
cli                                           ;Turn hardware int's back off
JMP     IBM_BIOS

```

```

;----- Menu CMD to test Console out code using this BIOS

```

```

MENU_CO_TEST:

```

```

        MOV     BX,CO_TEST_MSG           ;Keyboard test
        CALL    PRINT_STRING

```

```

Next_CO:MOV     BX,IN_CHAR_MSG           ;Input character =

```

```

        CALL    PRINT_STRING
        CALL    CI                       ;Return the char in AL

```

```

        CMP     AL,ESC

```

```

        JZ      CO_Done

```

```

        PUSH    AX

```

```

        MOV     CL,AL

```

```

        CALL    CO

```

```

MOV     BX,OUT_CHAR_MSG           ;"<---- Character recieved",CR,LF, Char displayed via Int 10H ="

```

```

CALL    PRINT_STRING

```

```

        POP     AX

```



```

MOV     AH,0EH                ;AH=0EH = TTY output, char in AL

int      10H                  ;Console out Handler   (Software Interrupt 10H)

CALL    CRLF

JMP      Next_CO

CO_Done:

cli                        ;Turn hardware int's back off

JMP      IBM_BIOS

;----- Menu CMD to test combined key-in / video out using this BIOS

MENU_BUFF_IO:

MOV      BX,BUFF_TEST_MSG    ;Keyboard buffer test

CALL     PRINT_STRING

mov      al,11111101b        ;Allow V1 on 8259A now

out      MASTER_PIC_PORT+1,al

sti                        ;Enable hardware interrupts

Next_CI:

```

```

MOV     AH,01H                ;Check if anything there
int     16H                   ;Get Keyboard status. Console Input Handler   (Software Interrupt 16H)
JZ      Next_CI

MOV     AH,0H                 ;Get actual character from buffer to AL
int     16H                   ;Get Character. Console Input Handler   (Software Interrupt 16H)

CMP     AL,ESC
JZ      CO_Done

MOV     AH,0EH                ;AH=0EH = TTY output, char in AL
int     10H                   ;Console out Handler   (Software Interrupt 10H)
JMP     Next_CI

```

```

;----- Menu CMD to test Serial Port character output using this BIOS to a serial terminal
;           Make sure you have the Baud rate is the same on both ends. (We will leave it at 38,4000 Baud)

```

```

MENU_SIO_TEST:

```

```

MOV     BX,SIO_TEST_MSG      ;Output to Serial port test
CALL    PRINT_STRING

MOV     AH,0H                ;AH=0 Initilize Port
MOV     AL,80H                ;This sets for 9600 Baud. However we will run at 38,400 (see INT 14H)
MOV     DX,0

```

```
int      14H                ;Serial out Handler   (Software Interrupt 14H)
```

```
OR       AH,AH              ;Any errors
```

```
JZ       Next_SIO
```

```
PUSH    AX
```

```
MOV     BX,SIO_INIT_ERR      ;Error initilizing Serial port
```

```
CALL    PRINT_STRING
```

```
POP     AX
```

```
MOV     AL,AH
```

```
CALL    AL_HEXOUT
```

```
MOV     BX,H_MSG_CRLF
```

```
CALL    PRINT_STRING
```

```
JMP     IBM_BIOS
```

```
Next_SIO:
```

```
CALL    CI                  ;Return the char in AL
```

```
CMP     AL,ESC
```

```
JZ      SIO_DoneTest
```

```
PUSH    AX
```

```
MOV     CL,AL
```

```
CALL    CO
```

```
POP     AX
```

```

MOV     AH,01H                ;AH=char output, char in AL
MOV     DX,0

      int      14H            ;Serial out Handler    (Software Interrupt 14H)

OR      AH,AH                 ;Any errors
JZ      Next_SIO
PUSH    AX
MOV     BX,SIO_ERR            ;Error sending to Serial port
CALL    PRINT_STRING
POP     AX
MOV     AL,AH
CALL    AL_HEXOUT
MOV     BX,H_MSG_CRLF
CALL    PRINT_STRING
JMP     IBM_BIOS

```

SIO_DoneTest:

```

      JMP     IBM_BIOS

```

;----- Menu CMD to test Timer code using this BIOS

MENU_TIMER_TEST:

```

MOV     BX,TIMER_TEST_MSG    ;Timer test

CALL    PRINT_STRING

mov     al,11111110b          ;Allow V0 on 8259A now

out     MASTER_PIC_PORT+1,al

sti                                           ;Enable hardware interrupts

```

Next_Timer:

```

MOV     BX,TIMER_DATA_MSG    ;Get Timer values

CALL    PRINT_STRING

CALL    CI                      ;Return the char in AL

CMP     AL,ESC

JZ      Timer_Done

MOV     BX,TIMER_LOW_MSG      ;timlow =

CALL    PRINT_STRING

PUSH    DS

XOR     AX,AX                  ;Set DS to data area for ROM usage in low RAM @ 400H....)

MOV     DS,AX

mov     AX,[timlow]

CALL    AX_HEXOUT

```

```

MOV     BX,TIMER_HIGH_MSG    ;timhi =
CALL    PRINT_STRING
mov     AX,[timhi]
CALL    AX_HEXOUT

```

```

MOV     BX,TIMER_OFLOW_MSG    ;timofl =
CALL    PRINT_STRING
mov     AX,[timofl]
CALL    AX_HEXOUT
MOV     BX,H_Msg              ;"H$"
CALL    PRINT_STRING
POP     DS
JMP     Next_Timer

```

Timer_Done:

```

mov     al,11111111b          ;Do not Allow V0 on 8259A again
out     MASTER_PIC_PORT+1,al
cli                                           ;Turn hardware int's back off
JMP     IBM_BIOS

```

```

;----- Menu CMD to test Floppy Disk (5") sequential sector reads using this BIOS
;
;      Will read sequentially up to 9 X 512 byte sectors from 5" DDDS 360K floppy (9 Sec/Track)
;
;      into RAM using the ZFDC controller board. (IBM says never more than
;
;      9 sectors at a time for this type of disk,  actually never changes the track #, but the
;
;      ZFDC can handle this if it did anyway!)

```

```
;          Will always read into RAM starting at 500H using the ZFDC controller board
```

```
FSEQ_5RD_TEST:
```

```
    PUSH    DS                ;Save Monitor current DS

    XOR     AX,AX              ;Set DS to data area for ROM usage in low RAM @ 400H....)

    MOV     DS,AX

                                ;
                                ;
                                ;

    MOV     BX,SEC_5RD_MSG      ;Say Reading sectors

    CALL    PRINT_STRING

                                ;
                                ;
                                ;

    MOV     AL,0H              ;Flag to indicate ZFDC board is NOT Initilized

    MOV     [ZFDC_INIT_FLAG],AL ;DS is already set for low RAM area

                                ;
                                ;
                                ;

    CALL    INIT_ZFDC          ;Initilize the ZFDC board hardware

                                ;
                                ;
                                ;

    CMP     byte [ZFDC_INIT_FLAG],0FFH ;Is Board initilized correctly

    JZ      ZFDC_5OK1

    POP     DS                ;Balance up Monitor stack

    JMP     IBM_BIOS
```

```
ZFDC_5OK1:
```

```
    mov     dl,01H            ;Drive 1, side A

    mov     ah,0H

                                ;
                                ;
                                ;

    int     13h               ;AH=0, reset floppy disk system
```

```

JNC     RESET_5OK1
MOV     BX,RESET_FAIL_MSG
CALL    PRINT_STRING
POP     DS                ;Balance up stack
JMP     IBM_BIOS

```

RESET_5OK1:

```

MOV     BX,SIDE_REQUEST_MSG
CALL    PRINT_STRING
call    CICO              ;Get a command from Console
PUSH    AX
MOV     BX,CRLFMSG        ;"CR,LF"
CALL    PRINT_STRING
POP     AX
CMP     AL,'B'
JZ      B5_SIDE
MOV     BX,SIDE_A_SET_MSG
CALL    PRINT_STRING
MOV     DX,0001H          ;Side A (DL bit 7 = 0 so Floppy disk)
JMP     OVER5_SIDE

```

B5_SIDE:MOV BX,SIDE_B_SET_MSG

```

CALL    PRINT_STRING
MOV     DX,0101H          ;Side B, Disk 1 (ZFDC #2)

```


OVER5_SIDE:

```

    PUSH    DX                      ;Save side info for below

    MOV     BX,ENTERRAM_FTRKL      ;"Track number, (xxH) "
    CALL    PRINT_STRING

    CALL    GET2DIGITS             ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)
    MOV     CH,AL

    PUSH    CX                      ;Save for below
    MOV     BX,H_MSG_CRLF         ;"H  CR,LF"
    CALL    PRINT_STRING

    MOV     BX,ENTERRAM_SECL      ;"Starting sector number, (xxH) = "
    CALL    PRINT_STRING

    CALL    GET2DIGITS             ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)
    POP     CX

    MOV     CL,AL

    PUSH    CX                      ;Save Track & Sec for below
    MOV     BX,H_MSG_CRLF         ;"H  CR,LF"
    CALL    PRINT_STRING

```

SEQ_50K4:

```

    MOV     BX,ENTER_COUNT        ;Enter # of sectors

    CALL    PRINT_STRING

    CALL    GET2DIGITS             ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)

```

```

        CMP     AL,09
        JBE     S5OK3
S5OK5: MOV     BX,OVER_COUNT_10
        CALL    PRINT_STRING
        JMP     SEQ_5OK4          ;Try again
S5OK3: OR      AL,AL
        JZ      S5OK5

        PUSH    AX                ;Save sector count (already in AL)
        MOV     BX,H_MSG_CRLF     ;"H  CR,LF"
        CALL    PRINT_STRING

        SUB     AX,AX
        MOV     ES,AX
        MOV     BX,500H           ;Will always dump data to 0000:500H
        POP     AX
        POP     CX                ;Track, Sec
        POP     DX                ;Side & Drive 0
mov      ah,02h                  ;Read x sectors  (IBM has a max of 15 sectors/call fot IBM-AT)
                                           ;on a 1.2M Floppy disk in their IBM PC-AT Bios. I assume 18 for 1.44 Disk)
                                           ;(This is where MS-DOS loads MSDOS.SYS from on disk)
        int     13H              ;AH=2, CX=0001, read 6 byte sectors -- as in early MSDOS systems!

        JNC     SEQ_5OK1          ;If NC then no errors
        MOV     BX,SQRD5FAILMSG

```

```

CALL    PRINT_STRING
POP     DS                      ;Balance up Monitor stack
JMP     IBM_BIOS                ;Will return back up to start of Monitor

```

SEQ_50K1:

```

MOV     BX,SQRD50KMSG
CALL    PRINT_STRING

MOV     CX,16                   ;Display the first 16 bytes at ES:BX in RAM
SUB     AX,AX
MOV     ES,AX
MOV     BX,500H                 ;Will always dump data to 0000:500H

CALL    SIMPLE_SECTOR_DUMP      ;Dump first CX bytes of sector data at ES:BX on CRT
POP     DS                      ;Balance up stack
JMP     IBM_BIOS                ;All done

```

```

;----- Menu CMD to test Floppy Disk (3") sequential sector reads using this BIOS
;
;      Will read sequentially 18 X 512 byte sectors from 3" DDDS 1.44M floppy (18 Sec/Track)
;
;      into RAM using the ZFDC controller board. (IBM says never more than
;
;      18 sectors at a time for this type of disk)
;
;      Will always read into RAM starting at 500H using the ZFDC controller board

```

FSEQ_3RD_TEST:

```

PUSH    DS                      ;Save Monitor current DS
XOR     AX,AX                   ;Set DS to data area for ROM usage in low RAM @ 400H....)
MOV     DS,AX

```

```

MOV     BX,SEC_3RD_MSG          ;Say Reading sectors
CALL    PRINT_STRING

```

```

MOV     AL,0H                   ;Flag to indicate ZFDC board is NOT Initilized
MOV     [ZFDC_INIT_FLAG],AL ;DS is already set for low RAM area

```

```

CALL    INIT_ZFDC               ;Initilize the ZFDC board hardware

```

```

CMP     byte [ZFDC_INIT_FLAG],0FFH ;Is Board initilized correctly
JZ      ZFDC_3OK1
POP     DS                      ;Balance up stack
JMP     IBM_BIOS

```

ZFDC_3OK1:

```

    mov     dl,0H                ;Drive 0, side A
    mov     ah,0H

    int     13H                  ;AH=0, reset floppy disk system

    JNC     RESET_3OK1
    MOV     BX,RESET_FAIL_MSG

```

```

CALL    PRINT_STRING

POP     DS                      ;Balance up stack

JMP     IBM_BIOS

```

RESET_3OK1:

```

MOV     BX,SIDE_REQUEST_MSG

CALL    PRINT_STRING

call    CICO                    ;Get a command from Console

PUSH    AX

MOV     BX,CRLFMSG              ;"CR,LF"

CALL    PRINT_STRING

POP     AX

CMP     AL,'B'

JZ      B3_SIDE

MOV     BX,SIDE_A_SET_MSG

CALL    PRINT_STRING

MOV     DX,0000H                ;Side A (DL bit 7 = 0 so Floppy disk)

JMP     OVER3_SIDE

```

B3_SIDE:MOV BX,SIDE_B_SET_MSG

```

CALL    PRINT_STRING

MOV     DX,0100H                ;Side B, Disk 0

```

OVER3_SIDE:

```

PUSH    DX                      ;Save side for below

```

```

MOV     BX,ENTERRAM_FTRKL    ;"Track number, (xxH) "
CALL    PRINT_STRING
CALL    GET2DIGITS           ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)
MOV     CH,AL
PUSH    CX                  ;Save for below
MOV     BX,H_MSG_CRLF        ;"H  CR,LF"
CALL    PRINT_STRING

```

```

MOV     BX,ENTERRAM_SECL     ;"Starting sector number, (xxH) = "
CALL    PRINT_STRING
CALL    GET2DIGITS           ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)
POP     CX
MOV     CL,AL
PUSH    CX                  ;Save Track & Sec for below
MOV     BX,H_MSG_CRLF        ;"H  CR,LF"
CALL    PRINT_STRING

```

SEQ_3OK4:

```

MOV     BX,ENTER_COUNT       ;Enter # of sectors
CALL    PRINT_STRING
CALL    GET2DIGITS           ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)
CMP     AL,18
JBE     S3OK3

```

S3OK5: MOV BX,OVER_COUNT_19

```

CALL    PRINT_STRING

JMP     SEQ_3OK4          ;Try again

S3OK3: OR     AL,AL

JZ      S3OK5

PUSH    AX                ;Save sector count (already in AL)

MOV     BX,H_MSG_CRLF     ;"H  CR,LF"

CALL    PRINT_STRING

SUB     AX,AX

MOV     ES,AX

MOV     BX,500H           ;Will always dump data to 0000:500H

POP     AX

POP     CX                ;Track, Sec

POP     DX                ;Side & Drive 0

mov     ah,02h            ;Read x sectors  (IBM has a max of 15 sectors/call fot IBM-AT)

                                ;on a 1.2M Floppy disk in their IBM PC-AT Bios. I assume 18 for 1.44 Disk)

                                ;(This is where MS-DOS loads MSDOS.SYS from on disk)

int     13H              ;AH=2, CX=0001, read 6 byte sectors -- as in early MSDOS systems!

JNC     SEQ_3OK1          ;If NC then no errors

MOV     BX,SQRD5FAILMSG

CALL    PRINT_STRING

POP     DS                ;Balance up stack

JMP     IBM_BIOS          ;Will return back up to start of Monitor

```

SEQ_3OK1:

```

MOV    BX,SQRD3OKMSG      ;Read sectors from 3" 1.44M Floppy disk OK
CALL   PRINT_STRING

MOV    CX,16              ;Display the first 16 bytes at ES:BX in RAM
SUB    AX,AX
MOV    ES,AX
MOV    BX,500H            ;Will always dump data to 0000:500H

CALL   SIMPLE_SECTOR_DUMP ;Dump first CX bytes of sector data at ES:BX on CRT
POP    DS                ;Balance up stack
JMP    IBM_BIOS           ;All done

```

;----- Menu CMD to test HDISK sequential sector READ's using this BIOS

```

;          Will read 512 byte sectors from 2nd IDE CF-Card
;          into RAM starting at 500H using the IDE controller board

```

HSEQ_RD_TEST:

```

PUSH    DS                ;Save Monitor current DS
XOR     AX,AX             ;Set DS to data area for ROM usage in low RAM @ 400H....)
MOV     DS,AX

MOV     BX,SEC_HDRD_MSG    ;Say Reading sectors

```



```

CALL    PRINT_STRING

MOV     BX,ONE_MOMENT_MSG    ;One moment while IDE disk is being initilized

CALL    PRINT_STRING


CALL    SET_DRIVE_B          ;Select the second Drive/CF card

CALL    IDEinit              ;Initialize drive 1. If there is no drive abort

JZ      HSEQ_RD1


MOV     BX,INIT_2_ERROR      ;Warn second IDE drive did not initilize

CALL    PRINT_STRING

POP     DS                   ;From above at start

JMP     IBM_BIOS             ;Will return back up to start of Monitor


HSEQ_RD1:

    mov     dl,80H           ;Hard Disk

    mov     ah,0H

    int     13h              ;AH=0, reset floppy disk system


JNC     HRESET_OK1

MOV     BX,HRESET_FAIL_MSG   ;"Reset of HDisk Failed"

CALL    PRINT_STRING

POP     DS                   ;From above at start

JMP     IBM_BIOS             ;Will return back up to start of Monitor

```

```

HRESET_OK1:

    XOR    AX,AX                ;Set DS to data area for ROM usage in low RAM @ 400H....)
    MOV    DS,AX

    MOV    BX,HRESET_OK_MSG    ;"Reset of HDisk OK"
    CALL   PRINT_STRING

AGAIN: MOV    BX,ENTERRAM_HEAD    ;"Starting HEAD number,(xxH) = "
    CALL   PRINT_STRING
    CALL   GET2DIGITS            ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)
    AND    AL,0FH
    MOV    [CURRENT_HEAD],AL
    MOV    BX,H_MSG_CRLF        ;"H  CR,LF"
    CALL   PRINT_STRING

    MOV    BX,ENTERRAM_FTRKL    ;"Track number,(xxH) "
    CALL   PRINT_STRING
    CALL   GET2DIGITS            ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)
    MOV    [CURRENT_TRACK],AL
    MOV    BX,H_MSG_CRLF        ;"H  CR,LF"
    CALL   PRINT_STRING

    MOV    BX,ENTERRAM_SECL     ;"Starting sector number,(xxH) = "
    CALL   PRINT_STRING
    CALL   GET2DIGITS            ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)

```

```

AND     AL,00111111B

MOV     [CURRENT_SECTOR],AL

MOV     BX,H_MSG_CRLF      ;"H  CR,LF"

CALL    PRINT_STRING

;-----

MOV     BX,ENTER_COUNT      ;Enter # of sectors

CALL    PRINT_STRING

CALL    GET2DIGITS          ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)

MOV     [SECTORS_TO_DO],AL

MOV     BX,H_MSG_CRLF      ;"H  CR,LF"

CALL    PRINT_STRING

;-----

MOV     BX,LOOP_ESC_MSG     ;"Will continously loop until ESC to abort "

CALL    PRINT_STRING

CALL    CI                  ;Wait for CR to start

CMP     AL,CR

JZ      XSEQ_50K4

POP     DS                  ;From above at start

JMP     IBM_BIOS            ;Will return back up to start of Monitor

XSEQ_50K4:

SUB     AX,AX

MOV     ES,AX

MOV     DS,AX

MOV     BX,500H            ;Will always dump data to 0000:500H

```

```

MOV     AH,02                ;Read sector(s)

MOV     AL,[SECTORS_TO_DO]

MOV     CH,[CURRENT_TRACK]

MOV     CL,[CURRENT_SECTOR]

MOV     DH,[CURRENT_HEAD]

MOV     DL,80H

INT     13H                  ;Disk I/O Int

JNC     READ_OK

JMP     RD_ERROR

READ_OK:

MOV     BX,SEC_READ_OK       ;Sector(s) read OK

CALL    PRINT_STRING

MOV     CX,16                ;Display the first 16 bytes at ES:BX in RAM

SUB     AX,AX

MOV     ES,AX

MOV     DS,AX                ;Just to be safe below also

MOV     BX,500H              ;Will always dump data to 0000:500H

CALL    SIMPLE_SECTOR_DUMP   ;Dump first CX bytes of sector data at ES:BX on CRT

CALL    CSTS                 ;Any keyboard character will stop display

```

```

JZ      HSEC_R7
CALL    CI
MOV     BX,CONTINUE_MSG
CALL    PRINT_STRING
CALL    CI
CMP     AL,ESC
JZ      IBM_BIOS1

```

HSEC_R7:

```

CALL    CRLF
MOV     CL,[CURRENT_SECTOR]
INC     CL
CMP     CL,DOS_MAXSEC      ;1-63 Sectors for custom Drive
JLE     R_SAME_HEAD
MOV     DH,[CURRENT_HEAD]
INC     DH
CMP     DH,DOS_MAXHEADS-1  ;(0...15), 16 heads Total for custom Drive
JLE     R_SAME_TRACK
MOV     byte [CURRENT_SECTOR],1    ;Back to sector 1
MOV     byte [CURRENT_HEAD],0      ;back to head 0
MOV     CH,[CURRENT_TRACK] ;Next track
INC     CH
MOV     [CURRENT_TRACK],CH
JMP     XSEQ_50K4            ;Do next sector block

```

R_SAME_TRACK:

```

MOV    byte [CURRENT_SECTOR],1    ;Back to sector 1
MOV    [CURRENT_HEAD],DH    ;Next head
JMP    XSEQ_50K4            ;Do next sector block

```

R_SAME_HEAD:

```

MOV    [CURRENT_SECTOR],CL
JMP    XSEQ_50K4            ;Do next sector block

```

IBM_BIOS1:

```

POP    DS
JMP    IBM_BIOS

```

RD_ERROR:

```

MOV    BX,RD_ERR_MSG        ;"Read Error Sector Head ="
CALL   PRINT_STRING
MOV    AL,[CURRENT_HEAD]
CALL   AL_HEXOUT
MOV    BX,TRACK_MSG         ;"H  Track ="
CALL   PRINT_STRING
MOV    AL,[CURRENT_TRACK]
CALL   AL_HEXOUT
MOV    BX,SEC_MSG           ;"H  Sector ="
CALL   PRINT_STRING
MOV    AL,[CURRENT_SECTOR]
CALL   AL_HEXOUT

```

```

MOV     BX,H_MSG_CRLF      ;"H  CR,LF"

CALL    PRINT_STRING

POP     DS                  ;Balance up stack

JMP     IBM_BIOS

```

```

;----- IBM Menu CMD to check Sector R/W functions on IDE Board using INT 13H.

```

```

HSEC_RW_TEST:

```

```

    PUSH    DS

    MOV     BX,HRW_TEST_MSG      ;Test Sector INT 13H Reade Write on Drive #2

    CALL    PRINT_STRING

    CALL    CICO

    CMP     AL,'Y'

    JZ      HSEC_RW0

    POP     DS                  ;Balance up stack

    JMP     IBM_BIOS

```

```

HSEC_RW0:

```

```

    MOV     BX,ONE_MOMENT_MSG    ;One moment while IDE disk is being initilized

    CALL    PRINT_STRING

```

```

CALL  SET_DRIVE_B          ;Select the second Drive/CF card
CALL  IDEinit              ;Initialize drive 1. If there is no drive abort
JZ     HSEC_RW1

MOV    BX,INIT_2_ERROR      ;Warn second IDE drive did not initilize
CALL  PRINT_STRING
POP    DS                  ;Balance up stack
JMP    IBM_BIOS

```

HSEC_RW1:

```

    mov    dl,80H           ;Reset Hard Disk
mov    ah,0H
    int    13h             ;AH=0, reset floppy disk system

```

```

JNC    HSEC_RW2
MOV    BX,HRESET_FAIL_MSG  ;"Reset of HDisk Failed"
CALL  PRINT_STRING
POP    DS                  ;From above at start
JMP    IBM_BIOS            ;Will return back up to start of Monitor

```

HSEC_RW2:

```

    XOR    AX,AX           ;Set DS to data area for ROM usage in low RAM @ 400H....)
    MOV    DS,AX

MOV    BX,HRESET_OK_MSG    ;"Reset of HDisk OK"

```



```
CALL PRINT_STRING
```

```
HSEC_RW3:
```

```
MOV BX,ENTERRAM_HEAD ;"Starting HEAD number,(xxH) = "
```

```
CALL PRINT_STRING
```

```
CALL GET2DIGITS ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)
```

```
AND AL,0FH
```

```
MOV [CURRENT_HEAD],AL
```

```
MOV BX,H_MSG_CRLF ;"H CR,LF"
```

```
CALL PRINT_STRING
```

```
MOV BX,ENTERRAM_FTRKL ;"Track number,(xxH) "
```

```
CALL PRINT_STRING
```

```
CALL GET2DIGITS ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)
```

```
MOV [CURRENT_TRACK],AL
```

```
MOV BX,H_MSG_CRLF ;"H CR,LF"
```

```
CALL PRINT_STRING
```

```
MOV BX,ENTERRAM_SECL ;"Starting sector number,(xxH) = "
```

```
CALL PRINT_STRING
```

```
CALL GET2DIGITS ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)
```

```
AND AL,00111111B
```

```
MOV [CURRENT_SECTOR],AL
```

```
MOV BX,H_MSG_CRLF ;"H CR,LF"
```

```
CALL PRINT_STRING
```

```

MOV     BX,ENTER_COUNT           ;Enter # of sectors
CALL    PRINT_STRING
CALL    GET2DIGITS               ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)
MOV     [SECTORS_TO_DO],AL
MOV     BX,H_MSG_CRLF           ;"H  CR,LF"
CALL    PRINT_STRING

MOV     BX,LOOP_ESC_MSG         ;"Will continously loop until ESC to abort "
CALL    PRINT_STRING
CALL    CI                      ;Wait for CR to start
CMP     AL,CR
JZ      HSEC_RW4
POP     DS                      ;From above at start
JMP     IBM_BIOS                ;Will return back up to start of Monitor

```

HSEC_RW4:

```

SUB     AX,AX
MOV     ES,AX
MOV     DS,AX                  ;just in case
MOV     BX,500H                ;Will always dump data to 0000:500H

MOV     AH,02                  ;Read sector(s)
MOV     AL,[SECTORS_TO_DO]
MOV     CH,[CURRENT_TRACK]

```

```
MOV    CL,[CURRENT_SECTOR]
```

```
MOV    DH,[CURRENT_HEAD]
```

```
MOV    DL,80H
```

```
INT     13H                ;Disk I/O Int
```

```
JNC     HSEC_RW5
```

```
JMP     RD_ERROR
```

```
HSEC_RW5:
```

```
MOV     CX,16              ;Display the first 16 bytes at ES:BX in RAM
```

```
SUB     AX,AX
```

```
MOV     ES,AX
```

```
MOV     DS,AX              ;Just in case
```

```
MOV     BX,500H            ;Will always dump data to 0000:500H
```

```
CALL    SIMPLE_SECTOR_DUMP ;Dump first CX bytes of sector data at ES:BX on CRT
```

```
SUB     AX,AX              ;Now WRITE the sector back
```

```
MOV     ES,AX
```

```
MOV     DS,AX              ;just in case
```

```
MOV     BX,500H            ;Will always dump data to 0000:500H
```

```
MOV     AH,03              ;Write sector(s)
```

```
MOV     AL,[SECTORS_TO_DO]
```

```

MOV     CH, [CURRENT_TRACK]
MOV     CL, [CURRENT_SECTOR]
MOV     DH, [CURRENT_HEAD]
MOV     DL, 80H

```

```

INT     13H                ;Write sector(s)

```

```

JNC     HSEC_RW6
JMP     WR_ERROR

```

HSEC_RW6:

```

MOV     BX, SEC_BACK_OK    ;Sector(s) written BACK OK
CALL    PRINT_STRING

CALL    CSTS                ;Any keyboard character will stop display
JZ      HSEC_RW7
CALL    CI
MOV     BX, CONTINUE_MSG
CALL    PRINT_STRING
CALL    CI
CMP     AL, ESC
JZ      IBM_BIOS2

```

HSEC_RW7:

```

CALL    CRLF
MOV     CL, [CURRENT_SECTOR]

```

```

INC     CL

CMP     CL,DOS_MAXSEC      ;1-63 Sectors for custom Drive

JLE     WR_SAME_HEAD

MOV     DH,[CURRENT_HEAD]

INC     DH

CMP     DH,DOS_MAXHEADS-1  ;(0...15), 16 heads Total for custom Drive

JLE     WR_SAME_TRACK

MOV     byte [CURRENT_SECTOR],1    ;Back to sector 1

MOV     byte [CURRENT_HEAD],0      ;back to head 0

MOV     CH,[CURRENT_TRACK] ;Next track

INC     CH

MOV     [CURRENT_TRACK],CH

JMP     HSEC_RW4              ;Do next sector block

```

WR_SAME_TRACK:

```

MOV     byte [CURRENT_SECTOR],1    ;Back to sector 1

MOV     [CURRENT_HEAD],DH    ;Next head

JMP     HSEC_RW4              ;Do next sector block

```

WR_SAME_HEAD:

```

MOV     [CURRENT_SECTOR],CL

JMP     HSEC_RW4              ;Do next sector block

```

IBM_BIOS2:

```

POP     DS

```

```
JMP    IBM_BIOS
```

```
WR_ERROR:
```

```
MOV     BX,WR_ERR_MSG      ;"Write Error Sector Head ="
```

```
CALL    PRINT_STRING
```

```
MOV     AL,[CURRENT_HEAD]
```

```
CALL    AL_HEXOUT
```

```
MOV     BX,TRACK_MSG       ;"H  Track ="
```

```
CALL    PRINT_STRING
```

```
MOV     AL,[CURRENT_TRACK]
```

```
CALL    AL_HEXOUT
```

```
MOV     BX,SEC_MSG         ;"H  Sector ="
```

```
CALL    PRINT_STRING
```

```
MOV     AL,[CURRENT_SECTOR]
```

```
CALL    AL_HEXOUT
```

```
MOV     BX,H_MSG_CRLF      ;"H  CR,LF"
```

```
CALL    PRINT_STRING
```

```
POP     DS                 ;Balance up stack
```

```
JMP     IBM_BIOS
```

```
;----- IBM Menu CMD to check HEX display / LBA selection on IDE Board.
```

```
;          Should show High Cylinder, Low Cylinder and Sector # in Hex Display on IDE Board
```

LBA_DISPLAY_TEST:

```

MOV     BX,LBA_TEST_MSG           ;Test LBA on Drive #2
CALL    PRINT_STRING

CALL    SET_DRIVE_B               ;Select the second Drive/CF card
CALL    IDEinit                   ;Initialize drive 1. If there is no drive abort
JZ      LBA_002

MOV     BX,INIT_2_ERROR           ;Warn second IDE drive did not initilize
CALL    PRINT_STRING
POP     DS
JMP     IBM_BIOS

```

LBA_002:

```

CALL    CRLF
MOV     DH,11100000B              ;<<<< Set to LBA mode, head 0
MOV     DL,REGshd                 ;Send "Head #" (in DH)
CALL    IDEwr8D                   ;Write to 8255 A Register

MOV     BX,TRKH_NUM               ;Enter High byte track number
CALL    PRINT_STRING
CALL    GET2DIGITS                ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)
MOV     DH,AL
MOV     DL,REGcylinderMSB
CALL    IDEwr8D                   ;Send High TRK# (in DH)

```

```

CALL  IDEwr8D_X          ;Special write to 8255 B Register (Not A) to update LED HEX Display
                                ;High 8 bits ignored by IDE drive

MOV   BX,TRKL_NUM        ;"Low Track number,(xxH)"
CALL  PRINT_STRING
CALL  GET2DIGITS         ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)
MOV   DH,AL              ;Get low Track #
MOV   DL,REGcylinderLSB  ;Send Low TRK# (in DH)
CALL  IDEwr8D            ;Special write to 8255 A

MOV   BX,SECTOR_NUM      ;"Sector number,(xxH) = "
CALL  PRINT_STRING
CALL  GET2DIGITS         ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)
MOV   DH,AL              ;Sector 1, Bits 0-5 only (currently 1-17)
MOV   DL,REGsector       ;Send "Sector#"
CALL  IDEwr8D            ;Write to 8255 A Register

MOV   AL,READcfg8255     ;Set 8255 back to read mode
OUT   IDECtrlPort,AL
MOV   BX,CHECK_DISPLAY_MSG ;Check display
CALL  PRINT_STRING
RET                        ;We arrive here from IDE Menu, return

```

```

;----- IBM Menu CMD to check HEX display CHS selection on IDE Board.

```


; Should show Cylinder, Head and Sector # in Hex Display on IDE Board

CHS_DISPLAY_TEST:

```

MOV     BX,CHS_TEST_MSG           ;Test CHS on Drive #2
CALL    PRINT_STRING

CALL    SET_DRIVE_B               ;Select the second Drive/CF card
CALL    IDEinit                   ;Initialize drive 1. If there is no drive abort
JZ      CHS_002

MOV     BX,INIT_2_ERROR           ;Warn second IDE drive did not initilize
CALL    PRINT_STRING
POP     DS
JMP     IBM_BIOS

```

CHS_002:

```

CALL    CRLF
OR      DH,10100000B              ;Set to >>>> NON-LBA mode <<<<
MOV     DL,REGshd                 ;Send "Head #" (in DH)
CALL    IDEwr8D                   ;Write to 8255 A Register

MOV     BX,TRKH_NUM               ;"Cylinder number High, (xxH)
CALL    PRINT_STRING
CALL    GET2DIGITS                 ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)

```

```

AND    AL,00000011B      ;Only 2 bits accepted
MOV     DH,AL
PUSH    AX                ;Save for below
MOV     DL,REGcylinderMSB
CALL    IDEwr8D           ;Send High TRK# (in DH) to IDE Drive

MOV     BX,HEAD_NUM       ;Enter Head number (0-FH)
CALL    PRINT_STRING
CALL    GET2DIGITS        ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)
MOV     DH,AL
AND     DH,0FH            ;top two LED HEX displays.
SHL     DH,1              ;These 8 (high) data lines are ignored by the IDE drive
SHL     DH,1
SHL     DH,1
SHL     DH,1
POP     AX                ;Get the tow bits of the high cylinder
OR      DH,AL
MOV     DL,REGcylinderMSB ;of the high cylinder in the low nibble.
CALL    IDEwr8D_X         ;Special output to 8255 B Register (Not A) to update LED HEX Display ONLY

MOV     BX,TRKL_NUM       ;"Low Cylinder number, (xxH) "
CALL    PRINT_STRING
CALL    GET2DIGITS        ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)
MOV     DH,AL             ;Get low Track #
MOV     DL,REGcylinderLSB ;Send Low TRK# (in DH)

```

```

CALL    IDEwr8D                ;Special write to 8255 A

MOV     BX,SECTOR_NUM          ;"Sector number,(xxH) = "
CALL    PRINT_STRING

CALL    GET2DIGITS              ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)
MOV     DH,AL                   ;Sector 1, Bits 0-5 only  (currently 1-17)
MOV     DL,REGsector            ;Send "Sector#"
CALL    IDEwr8D                ;Write to 8255 A Register

MOV     AL,READcfg8255          ;Set 8255 back to read mode
OUT     IDECtrlPort,AL

MOV     BX,CHECK_DISPLAY_MSG    ;Check display
CALL    PRINT_STRING

JMP     IBM_BIOS                ;Will return back up to start IBM Menu

```

```

;----- Menu command to dump a floppy BOOT sector on the CRT

```

```

;           Note must have a functional INT 13H routine for this section to work

```

```

DUMP_B_SEC:

```

```

    PUSH    DS                  ;Save Monitor current DS

    XOR     AX,AX               ;Set DS to data area for ROM usage in low RAM (400H))

    MOV     DS,AX

```

```

MOV     BX,BOOT_3RD_MSG           ;Say Reading Boot sector
CALL    PRINT_STRING

MOV     AL,0H                     ;Flag to indicate ZFDC board is NOT Initilized
MOV     [ZFDC_INIT_FLAG],AL       ;DS is already set for low RAM area

CALL    INIT_ZFDC                 ;Initilize the ZFDC board hardware

CMP     byte [ZFDC_INIT_FLAG],0FFH ;Is Board initilized correctly
JZ      BS_ZFDC_3OK1

pop     ds                        ;Balance up Monitor stack
JMP     IBM_BIOS

```

BS_ZFDC_3OK1:

```

MOV     BX,DRIVE_SELECT_MSG ;Floppy disk A: or B:
CALL    PRINT_STRING

call    CICO                      ;Get a command from Console
PUSH    AX

MOV     BX,CRLFMSG                ;"CR,LF"
CALL    PRINT_STRING

POP     AX

CMP     AL,'B'
JZ      B_DRIVE_SEL

```

```

        MOV     DX,0000H           ;Side A, Disk 0

        JMP     OVER_DRIVE_SEL

B_DRIVE_SEL:

        MOV     DX,00001H         ;Side A, Disk 1

OVER_DRIVE_SEL:

        PUSH    DX                ;Save side for below


        mov     ah,0H

        int     13h               ;AH=0, reset floppy disk system


        JNC     BS_RESET_3OK1

        MOV     BX,RESET_FAIL_MSG

        CALL    PRINT_STRING

        pop     dx

        pop     ds                ;Balance up stack

        JMP     IBM_BIOS


BS_RESET_3OK1:

        SUB     AX,AX

        MOV     ES,AX

        MOV     BX,500H           ;Will always dump data to 0000:500H


        POP     DX                ;Side & Drive 0

        MOV     CX,0001           ;1st sector on track 0

```

```

MOV     AL,1                ;1 sector
mov     ah,02h              ;Read 1 sector

int     13H

JNC     BS_SEQ_3OK1         ;If NC then no errors
MOV     BX,BOOT_INFO_FAIL_MSG
pop     ds                  ;Balance up Monitor stack
JMP     IBM_BIOS            ;Will return back up to start of Monitor

```

BS_SEQ_3OK1:

```

MOV     BX,BOOT_INFOOKMSG
CALL    PRINT_STRING

SUB     AX,AX
MOV     DS,AX
MOV     SI,500H             ;Will always dump data to 0000:500H

LODSB                      ;WRITE 1 BYTE BYTE, DS:[SI++] -> AL
CALL    AL_HEXOUT
LODSB
CALL    AL_HEXOUT
LODSB
CALL    AL_HEXOUT
MOV     BX,JMP_MSG         ;" BOOT JUMP VECTOR"

```

```

CALL    PRINT_STRING

MOV     DL,8

BS_1:   LODSB                      ;Get a byte from RAM, DS:[SI++] -> AL
MOV     CL,AL
CALL    CO
DEC     DL
JNZ     BS_1
MOV     BX,NAME_MSG                ;"   OEM NAME"
CALL    PRINT_STRING

LODSW
CALL    AX_HEXOUT
MOV     BX,BYTES_MSG               ;"   Bytes/Sec"
CALL    PRINT_STRING
LODSB
CALL    AL_HEXOUT
MOV     BX,CLUSTER_MSG             ;"   Sec/Cluster"
CALL    PRINT_STRING
LODSW
CALL    AX_HEXOUT
MOV     BX,RES_MSG                 ;"   Reserved Sectors"
CALL    PRINT_STRING
LODSB
CALL    AL_HEXOUT

```

```

MOV     BX,FATS_MSG           ;"   FATS"
CALL    PRINT_STRING
LODSW
CALL    AX_HEXOUT
MOV     BX,ROOT_MSG           ;"   Root Dir Entries"
CALL    PRINT_STRING
LODSW
CALL    AX_HEXOUT
MOV     BX,SECTORS_MSG         ;"   Sectors"
CALL    PRINT_STRING
LODSB
CALL    AL_HEXOUT
MOV     BX,MEDIA_MSG           ;"   Media Byte"
CALL    PRINT_STRING
LODSW
CALL    AX_HEXOUT
MOV     BX,FAT_SEC_MSG         ;"   FAT Sectors"
CALL    PRINT_STRING
LODSW
CALL    AX_HEXOUT
MOV     BX,SEC_TRK_MSG         ;"   Sectors/Track"
CALL    PRINT_STRING
LODSW
CALL    AX_HEXOUT
MOV     BX,HEADS_MSG           ;"   Heads"

```



```
CALL PRINT_STRING
LODSW
CALL AX_HEXOUT
LODSW
CALL AX_HEXOUT
MOV BX,HIDDEN_MSG ;" Hidden Sectors"
CALL PRINT_STRING
LODSW
CALL AX_HEXOUT
LODSW
CALL AX_HEXOUT
MOV BX,HUGE_MSG ;" Huge Sectors"
CALL PRINT_STRING
LODSB
CALL AL_HEXOUT
MOV BX,DRIVE_NO_MSG ;" Drive #"
CALL PRINT_STRING
LODSB
CALL AL_HEXOUT
MOV BX,RESERVED_MSG ;" Reserved"
CALL PRINT_STRING
LODSB
CALL AL_HEXOUT
MOV BX,BOOT_SIG_MSG ;" Boot Signature"
CALL PRINT_STRING
```



```
JMP    IBM_BIOS            ;All done
```

```
;
```

```
;----- Menu command to dump the Hard Disk MBR (Master Boot Record) Info on the CRT
```

```
;          Note must have a functional INT 13H routine for this section to work
```

```
DUMP_MBR:
```

```
    PUSH    DS                ;Save Monitor current DS
```

```
    XOR     AX,AX              ;Set DS to data area for ROM usage in low RAM (400H))
```

```
    MOV     DS,AX
```

```
    MOV     BX,BOOT_MBR_MSG    ;Say Reading MBR sector
```

```
    CALL    PRINT_STRING
```

```
    CALL    SET_DRIVE_B        ;Select the second Drive/CF card
```

```
    CALL    IDEinit            ;Initialize drive 1. If there is no drive abort
```

```
    JZ      MBR_002
```

```
    MOV     BX,INIT_2_ERROR     ;Warn second IDE drive did not initilize
```

```
    CALL    PRINT_STRING
```

```
    POP     DS
```

```
    JMP     IBM_BIOS
```

```
MBR_002:
```

```

SUB    AX,AX
MOV    ES,AX
MOV    BX,500H                ;Will always dump data to 0000:500H

```

```

MOV    DX,0080H              ;Head 0, HDIdk 0
MOV    CX,0001                ;1st sector on track 0
MOV    AL,1                   ;read 1 sector
mov     ah,02h                ;Read 1 sector

```

```

int     13H

```

```

JNC    MBR_003                ;If NC then no errors
MOV    BX,BOOT_MBR_FAIL_MSG
pop     ds                    ;Balance up Monitor stack
JMP     IBM_BIOS              ;Will return back up to start of Monitor

```

```

MBR_003:

```

```

MOV    BX,MBR_INFOOKMSG
CALL   PRINT_STRING

```

```

SUB    AX,AX
MOV    DS,AX
MOV    SI,500H                + 1B8H    ;Will always dump data to 0000:500H

```

```

LODSB                                ;WRITE 1 BYTE BYTE, DS:[SI++] -> AL

```

```

CALL    AL_HEXOUT
LODSB
CALL    AL_HEXOUT
LODSB
CALL    AL_HEXOUT
LODSB
CALL    AL_HEXOUT
MOV     BX,DISK_SIG_MSG           ;"  Disk Signature (Optional)"
CALL    PRINT_STRING

LODSB
CALL    AL_HEXOUT
LODSB
CALL    AL_HEXOUT
MOV     BX,NULS_MSG              ;"  Usually Nulls (Optional)"
CALL    PRINT_STRING

LODSB                             ;0
CALL    AL_HEXOUT
MOV     BX,STATUS_MSG           ;"          Status Byte"
CALL    PRINT_STRING
MOV     BX,PT1_MSG              ;"First Partition Table "
CALL    PRINT_STRING
CALL    DUMP_PTBL

```

```

LODSB                ;0

CALL    AL_HEXOUT

MOV     BX,STATUS_MSG ;"          Status Byte"

CALL    PRINT_STRING

MOV     BX,PT2_MSG    ;"Second Partition Table "

CALL    PRINT_STRING

CALL    DUMP_PTBL


LODSB                ;0

CALL    AL_HEXOUT

MOV     BX,STATUS_MSG ;"          Status Byte"

CALL    PRINT_STRING

MOV     BX,PT3_MSG    ;"Third Partition Table "

CALL    PRINT_STRING

CALL    DUMP_PTBL


LODSB                ;0

CALL    AL_HEXOUT

MOV     BX,STATUS_MSG ;"          Status Byte"

CALL    PRINT_STRING

MOV     BX,PT4_MSG    ;"Forth Partition Table "

CALL    PRINT_STRING

CALL    DUMP_PTBL


LODSW

```



```
LODSB
CALL  AL_HEXOUT
LODSB
CALL  AL_HEXOUT
MOV   BX,ECHS_MSG      ;"          End CHS Address"
CALL  PRINT_STRING

LODSW          ;8-B
CALL  AX_HEXOUT
LODSW
CALL  AX_HEXOUT
MOV   BX,SLB_MSG       ;"          Start LBA Address"
CALL  PRINT_STRING

LODSW          ;C-F
CALL  AX_HEXOUT
LODSW
CALL  AX_HEXOUT
MOV   BX,ELBA_MSG      ;"          End LBA Address"
CALL  PRINT_STRING
RET
```



```
;----- Menu CMD to turn on/off BIOS dump info for reads/writes using this BIOS
```

```
;   DEBUG_FLAG = 0      if no debugging info sent to serial terminal
```

```
;   DEBUG_FLAG = 1      send just INT's info
```

```
;   DEBUG_FLAG = 2      send more detailed information
```

```
DEBUG_ON_OFF:
```

```
    push    ds
```

```
    XOR     AX,AX                ;Set DS to data area for ROM usage in low RAM @ 400H....)
```

```
    MOV     DS,AX
```

```
    MOV     BX,DEBUG_SET_MSG
```

```
    CALL    PRINT_STRING
```

```
    call    CICO                ;Look for 0,1 2 (only)
```

```
    CMP     AL,'1'
```

```
    MOV     byte [DEBUG_FLAG],01H
```

```
    MOV     BX,DUMP_ON1_MSG
```

```
    JZ      DUMP_DONE
```

```
    CMP     AL,'2'
```

```
    MOV     byte [DEBUG_FLAG],02H
```

```
    MOV     BX,DUMP_ON2_MSG
```

```
    JZ      DUMP_DONE
```

```
    CMP     AL,'3'
```

```
MOV    byte [DEBUG_FLAG],03H
```

```
MOV    BX,DUMP_ON3_MSG
```

```
JZ     DUMP_DONE
```

```
MOV    byte [DEBUG_FLAG],0
```

```
MOV    BX,DUMP_OFF_MSG
```

```
DUMP_DONE:
```

```
CALL   PRINT_STRING
```

```
POP    DS
```

```
RET
```

```
;*****
```

```
;
```

```
;      Bootstrap Handler      (IBM-PC Software Interrupt 19H)
```

```
;
```

```
;      SYSTEM - BOOTSTRAP LOADER
```

```
;
```

```
;      For a floppy the BIOS will try to read sector 1, head 0, track 0 from drive A:
```

```
;      to 0000h:7C00h.  If this fails we will just abort.
```

```
;      For the IDE/CF Cards the BIOS will try to read sector 1, head 0, track 0 from
```

```
;      drive #2 of the IDE Board to 0000h:7C00h.  If this fails we will just abort.
```

```

;

;   For a hard disk, the BIOS will read sector 1, head 0, track 0 of the 2nd CF-Card
;   on the Dual IDE board. This sector should contain a master bootstrap loader and
;   a partition table (see http://www.ctyme.com/intr/rb-2270.htm#Table650).
;

;   After loading the master boot sector at 0000h:7C00h,
;   the master bootstrap loader is given control with:-
;

;           CS:IP = 0000h:7C00h.
;
;           DH = access bits 7-6,4-0: Don't care
;           bit 5:=0 device supported by INT 13.
;
;           DL = boot drive
;
;                   00h first floppy
;                   80h first hard disk
;

;   True IBM PCs and most clones issue an INT 18 (cassette) if neither floppy nor hard
;   disk have a valid boot sector. We will just abort.
;

;   To accomplish a warm boot equivalent to Ctrl-Alt-Del, store 1234h in
;   0040h:0072h and jump to FFFF:0000h. For a cold boot equivalent to
;   a reset, store 0000h at 0040h:0072h before jumping..
;

;   BUG: If when loading the remainder of the DOS system files fails, various versions
;   of IBMBIO.COM/IO.SYS incorrectly restore INT 1E before calling INT 19, assuming
;   that the boot sector had stored the contents of INT 1E at DS:SI instead of on

```

```
; the stack as it actually does.
```

```
;*****
```

```
BOOT_DOS_INT:
```

```
    STI                      ;Bootstrap Handler (Interrupt 19H)

    CMP    DL,80H            ;Floppy or HD? (80H = First HD)

    JNZ    BOOT_FLOPPY

    JMP    BOOT_HDISK        ;Z then BOOT FLOPPY. Note if called, ZFDC Board MUST be active (IDE may be offline)
```

```
BOOT_FLOPPY:
```

```
    PUSH    DS                ;Save current DS on stack

    XOR     AX,AX             ;Set DS to data area for ROM usage in low RAM @ 400H....)

    MOV     DS,AX

                                ;Flag to indicate ZFDC board is NOT Initilized

    MOV     AL,0H

    MOV     [ZFDC_INIT_FLAG],AL

                                ;Initilize the ZFDC board hardware (360K & 1.44M disks)

    CALL    INIT_ZFDC

                                ;Is Board initilized correctly

    CMP     byte [ZFDC_INIT_FLAG],0FFH

    POP     DS                ;Balance up stack

    JZ      ZFDC_OK

    JMP     IBM_BIOS          ;Return will drop back to IBM_BIOS location
```

```

ZFDC_OK:

    CALL    SET_DRIVE_B          ;Select the second Drive/CF card

    CALL    IDEinit              ;Initialize drive 1. If there is no drive abort

    JZ      FH_RESET_OK

    MOV     BX,INIT_2_ERROR      ;Warn second IDE drive did not initilize

    CALL    PRINT_STRING        ;Continue anyway with ZFDC/Floppy

FH_RESET_OK:

    sub     ax,ax

    mov     ds,ax               ;DS -> 0

    mov     dx,0080H            ;DL = 80L will always boot from IDE #2 disk for now.

    mov     ah,0

    int     13h                ;AH=0, reset floppy disk system

    JNC     F_RESET_OK

    MOV     BX,RESET_FAIL_MSG

    CALL    PRINT_STRING

    JMP     IBM_BIOS            ;Will return back up to IBM_BIOS location

F_RESET_OK:

    XOR     AX,AX

    mov     DS,AX

```

```

mov     ES,AX                ;DS = ES = 0000H

mov     ax,201h              ;read one sector

mov     bx,DOS_BOOT_LOC      ;set ES:BX to data destination 7C00H (BB,00,7c)

mov     cx,0001H             ;Track 0, sec 01

mov     dx,0000H             ;side A, (DL bit 7 = 0) drive 0,

int     13H                  ;AH=2, CX=1, read 1 (the boot), sector

JNC     F_BOOT_OK            ;If NC, then no errors

MOV     BX,BOOT_FAIL_MSG

CALL    PRINT_STRING

JMP     IBM_BIOS              ;Will return back up to IBM_BIOS location

```

F_BOOT_OK:

```

XOR     AX,AX

MOV     DS,AX

CMP     word [DOS_BOOT_SIGNATURE],0AA55H ;Check we have a valid MBL signature

JZ      F_BOOT_OK1

MOV     BX,NO_MBL_MSG        ;No Floppy Boot Loader Signature detected

CALL    PRINT_STRING

JMP     IBM_BIOS              ;Will return back up to IBM_BIOS location

```

F_BOOT_OK1:

```

MOV     BX,BOOT_OK_MSG

```

```

CALL    PRINT_STRING

;      Call    CI                      ;Wait for CRT input for boot debugging  (info at 7C00H)

MOV     DX,0                          ;Required see above

JMP     word  0000H:DOS_BOOT_LOC ;Far Jump, execute the boot code @0:7C00H


                                ;<<< BOOT HDISK .  IDE  Board MUST be active (ZFDC board may be offline)
BOOT_HDISK:                      ;Boot MSDOS (or FreeDOS) from IDE/CF Card

CALL    SET_DRIVE_B                ;Select the second Drive/CF card

CALL    IDEinit                    ;Initialize drive 1. If there is no drive abort

JZ      BOOT_RESET_OK

MOV     BX,INIT_2_ERROR              ;Warn second IDE drive did not initilize

CALL    PRINT_STRING

JMP     IBM_BIOS                    ;Will return back up to start of Monitor


BOOT_RESET_OK:

PUSH    DS

XOR     AX,AX                        ;Set DS to data area for ROM usage in low RAM @ 400H....)

MOV     DS,AX

MOV     AL,0H                        ;Flag to indicate ZFDC board is NOT Initilized

MOV     [ZFDC_INIT_FLAG],AL

```

```

CALL    INIT_ZFDC                ;Initilize the ZFDC board hardware (360K & 1.44M disks)

POP     DS                      ;Balance up stack

```

BOOT_ZFDC_OK:

```

sub     ax,ax

mov     ds,ax                  ;DS -> 0

mov     dx,0080H              ;DL = 80H  will always boot from HDisk #2


mov     ah,0

int     13H                   ;AH=0, reset Hard Disk system


JNC     HBOOT_RESET_OK

MOV     BX,RESET_FAIL_MSG

CALL    PRINT_STRING

JMP     IBM_BIOS               ;Will return back up to IBM_BIOS location

```

HBOOT_RESET_OK:

```

XOR     AX,AX

mov     DS,AX

mov     ES,AX                  ;DS = ES = 0000H

mov     ax,201h                ;read one sector

mov     bx,DOS_BOOT_LOC        ;set ES:BX to data destination 7C00H (BB,00,7c)

mov     cx,0001H              ;Track 0, sec 01 for MBL >>> Boot on Sector 12H <<<

mov     dx,0080H              ;head 0, HDisk 0, (DL bit 7 = 1)

```



```

        int      13H                ;AH=2, CX=1, read 1 (the boot), sector

JNC     HDOS_BOOT_OK                ;If NC, then no errors
MOV     BX,BOOT_FAIL_MSG
CALL    PRINT_STRING
JMP     IBM_BIOS                    ;Will return back up to IBM_BIOS location

HDOS_BOOT_OK:
        MOV     BX,BOOT_OK_MSG
        CALL    PRINT_STRING

;      Call    CI                    ;Wait for CRT input for boot debugging
                                         ; (Can reset and look at BOOT sector)

        MOV     DX,0080H            ;Required see above
        JMP     word 0000H:DOS_BOOT_LOC ;Far Jump, execute the boot code @0:7C00H

;*****

;

;      Disk I/O Handler            (Software Interrupt 13H & 40H)
;

;Input:      AH = 00h            DISK - RESET DISK SYSTEM
;      DL = drive (if bit 7 is set, both hard disks and floppy disks are reset)

```

```

;Return:AH = status (see below)

;      CF clear if successful (returned AH=00h)

;      CF set on error

;

;

;Input:      AH = 01h      DISK - GET STATUS OF LAST OPERATION

;      DL = drive (bit 7 set for hard disk)

;Return:CF clear if successful (returned status 00h)

;      CF set on error

;      AH = status of previous operation (see below)

;

;

;Input:      AH = 02h      READ  03H,WRITE SECTOR DATA

;      AL = number of sectors to read (must be nonzero)

;      CH = low eight bits of cylinder number

;      CL = sector number 1-63 (bits 0-5, high two bits of cylinder (bits 6-7, hard disk only)

;      DH = head number

;      DL = drive number (bit 7 set, for hard disk)

;      ES:BX -> data buffer

;

;Return:CF set on error

;      if AH = 11h (corrected ECC error), AL = burst length

;      CF clear if successful

;      AH = status (see below)

;      AL = number of sectors transferred (only valid if CF set for some BIOSes)

```

```

;
;
;Input:      AH = 04h      DISK - VERIFY DISK SECTOR(S)
;
;   AL = number of sectors to verify (must be nonzero)
;
;   CH = low eight bits of cylinder number
;
;   CL = sector number 1-63 (bits 0-5) high two bits of cylinder (bits 6-7, hard disk only)
;
;   DH = head number
;
;   DL = drive number (bit 7 set, for hard disk)
;
;   ES:BX -> data buffer (PC,XT,AT with BIOS prior to 1985/11/15)
;
;Return:CF set on error
;
;   CF clear if successful
;
;   AH = status (see below)
;
;   AL = number of sectors verified
;
;
;
;Input:      AH = 05h      FLOPPY - FORMAT TRACK
;
;   AL = number of sectors to format
;
;   CH = track number
;
;   DH = head number
;
;   DL = drive number
;
;   ES:BX -> address field buffer:-
;
;           00h    BYTE    track number
;
;           01h    BYTE    head number (0-based)
;
;           02h    BYTE    sector number

```

```

;          03h    BYTE    sector size (00h=128 bytes, 01h=256 bytes, 02h=512, 03h=1024)

;Return:CF set on error

;    CF clear if successful

;    AH = status (see below)

;          Note: On AT or higher, call AH=17h first. The number of sectors per track is read from the diskette
;          parameter table pointed at by INT 1E

;

;Input:    AH = 08h    RETURN DRIVE PARAMATERS

;    DL = drive number (bit 7 set, for hard disk)

;    ES:DI = 0000H:0000H

;          Note: For systems predating the IBM AT, this call is only valid for hard disks, as it is implemented
;          by the hard disk BIOS rather than the ROM BIOS. The IBM ROM-BIOS returns the total number of hard disks
;          attached to the system regardless of whether DL >= 80h on entry

;Return:CF set on error

;    CF clear if successful

;    AH = status (see below)

;    AL = 00h on at least some BIOSes

;    BL = drive type (AT/PS2 floppies only)

;          Values for diskette drive type:

;          01h    360K

;          02h    1.2M

;          03h    720K

;          04h    1.44M

;    CH = low eight bits of maximum cylinder number

```

```

;      CL = maximum sector number (bits 5-0)

;      high two bits of maximum cylinder number (bits 7-6)

;      DH = maximum head number

;      DL = number of drives

;      ES:DI -> drive parameter table (floppies only)


;Input:      AH = 15h      GET DISK TYPE

;      DL = drive number (bit 7 set, for hard disk)


;Return:CF set on error

;      CF clear if successful

;      AH = type code (see below)

;      00h no such drive

;      01h floppy without change-line support

;      02h floppy (or other removable drive) with change-line support

;      03h hard disk

;      CX:DX = number of 512-byte sectors


;RETURNED ERROR CODES IN AH:-

;      00h      successful completion

;      01h      invalid function in AH or invalid parameter

;      02h      address mark not found

;      03h      disk write-protected

;      04h      sector not found/read error

;      05h      reset failed (hard disk)

```

```
; 05h    data did not verify correctly (TI Professional PC)
; 06h    disk changed (floppy)
; 07h    drive parameter activity failed (hard disk)
; 08h    DMA overrun
; 09h    data boundary error (attempted DMA across 64K boundary or >80h sectors)
; 0Ah    bad sector detected (hard disk)
; 0Bh    bad track detected (hard disk)
; 0Ch    unsupported track or invalid media
; 0Dh    invalid number of sectors on format (PS/2 hard disk)
; 0Eh    control data address mark detected (hard disk)
; 0Fh    DMA arbitration level out of range (hard disk)
; 10h    uncorrectable CRC or ECC error on read
; 11h    data ECC corrected (hard disk)
; 20h    controller failure
; 31h    no media in drive (IBM/MS INT 13 extensions)
; 32h    incorrect drive type stored in CMOS (Compaq)
; 40h    seek failed
; 80h    timeout (not ready)
; AAh    drive not ready (hard disk)
; B0h    volume not locked in drive (INT 13 extensions)
; B1h    volume locked in drive (INT 13 extensions)
; B2h    volume not removable (INT 13 extensions)
; B3h    volume in use (INT 13 extensions)
; B4h    lock count exceeded (INT 13 extensions)
; B5h    valid eject request failed (INT 13 extensions)
```

```
; B6h    volume present but read protected (INT 13 extensions)
; BBh    undefined error (hard disk)
; CCh    write fault (hard disk)
; E0h    status register error (hard disk)
; FFh    sense operation failed (hard disk)
```

```
;*****
```

```
OLD_DISKIO:                ;Come here via INT 40H, (rearily) for the old Floppy Disk relocated INTs
    STI                    ;Normal INT 13H Entry point
    PUSH    DS              ;For all commands use variables in low RAM if needed
    PUSH    AX
    XOR     AX,AX           ;Set DS to data area for ROM usage in low RAM @ 400H....)
    MOV     DS,AX
    POP     AX

    CMP     byte [DEBUG_FLAG],0    ;Is Floppy Debug mode on
    JZ      COMMON_DISK_COMMANDS   ;If not skip to "normal" FDisk routines

    PUSH    AX
    PUSH    BX
    PUSH    CX
    MOV     BX,INT_40F_MSG          ;"Int 40H (<--Floppy) AX="
    CALL    SERIAL_PRINT_STRING
    POP     CX
```

```

    POP    BX

    POP    AX

    CALL   SERIAL_DISPLAY_REGISTERS ;Display Registers on serial port display (All registers retained)

    JMP    COMMON_DISK_COMMANDS    ;Go to "normal" Disk routines

DISKIO:                                     ;Normal INT 13H Entry point

    STI

    PUSH   DS                            ;For all commands use variables in low RAM if needed

    PUSH   AX

    XOR    AX,AX                        ;Set DS to data area for ROM usage in low RAM @ 400H....)

    MOV    DS,AX

    POP    AX

    CMP    byte [DEBUG_FLAG],0          ;Is Debug mode on

    JZ     COMMON_DISK_COMMANDS        ;If not skip

    PUSH   AX

    PUSH   BX

    PUSH   CX

    TEST   DL,80H                       ;Floppy or HDisk

    JNZ    DISKIO1

    MOV    BX,INT_13F_MSG                ;"Int 13H (Floppy) AX="

    JMP    DISKIO2

DISKIO1:MOV    BX,INT_13H_MSG            ;"Int 13H (** HDisk **) AX="

DISKIO2:CALL   SERIAL_PRINT_STRING

```



```

POP    CX

POP    BX

POP    AX

CALL   SERIAL_DISPLAY_REGISTERS ;Display Registers on serial port display (All registers retained)
                                           ;Fall through to COMMON_DISK_COMMANDS

```

COMMON_DISK_COMMANDS:

```

TEST   DL,80H                      ;HDisk or Floppy CMD

JZ     FD_COMMANDS                 ;For Floppy disk commands

JMP    HD_COMMANDS                 ;For HDISK Commands

```

;----- Floppy Disk Commands -----

FD_COMMANDS:

```

TEST   AH,AH                      ;Is it a FDisk reset

JNZ    N_FDISK_RESET

JMP    FDISK_RESET

```

N_FDISK_RESET:

```

CMP     AH,1                      ;Is it a FDisk status request

JNZ     N_FDISK_STATUS

JMP     FDISK_STATUS

```

N_FDISK_STATUS:

```

CMP     AH,2                      ;Is it a FDisk read request

JNZ     N_FDISK_READ

MOV     byte [VERIFY_FLAG],0H      ;Turn off verify flag

```

```

        JMP     FDISK_READ

N_FDISK_READ:
        CMP     AH,3                      ;Is it a FDisk write request
        JNZ     N_FDISK_WRITE
        JMP     FDISK_WRITE

N_FDISK_WRITE:
        CMP     AH,4                      ;Is it a FDisk Verify request
        JNZ     N_FDISK_VERIFY
        MOV     byte [VERIFY_FLAG],0ffH    ;Turn on verify flag
        JMP     FDISK_READ                ;Modified read

N_FDISK_VERIFY:
        CMP     AH,5                      ;Is it a FDisk format request
        JNZ     N_FDISK_FORMAT
        JMP     FDISK_FORMAT

N_FDISK_FORMAT:
        CMP     AH,8                      ;Is it a FDisk paramaters request
        JNZ     N_FDISK_PARAMS
        JMP     FDISK_PARAMS

N_FDISK_PARAMS:
        CMP     AH,15H                    ;GET DISK TYPE (XT 1986/1/10 or later,XT286,AT,PS)
        JNZ     N_FDISK_DASB
        JMP     FDISK_DASB

N_FDISK_DASB:
        CMP     AH,16H                    ;FDisk media change check request
        JNZ     NOT_VALID_DISK

```

```
JMP    FDISK_MEDIA_CHANGE
```

```
NOT_VALID_DISK:                ;Thats all for now
```

```
PUSH    AX
```

```
MOV     BX,INVALID_AH_FMSG
```

```
CALL    PRINT_STRING
```

```
POP     AX
```

```
MOV     AL,AH
```

```
CALL    AL_HEXOUT
```

```
MOV     BX,H_MSG_CRLF
```

```
CALL    PRINT_STRING
```

```
mov     byte [IBM_DISK_STATUS],cmderr    ;Show bad command
```

```
                ;Fall through to DONE_DISK
```

```
DONE_DISK:                    ;Most (but not all), floppy commands come back here before returning to DOS
```

```
mov     ah,[IBM_DISK_STATUS]
```

```
OR      AH,AH                ;Was there an error
```

```
JZ      ALL_OK
```

```
STC                                ;Set carry to indicate an error
```

```
ALL_OK:
```

```
POP     ds                    ;Get back the original saved DS at start
```

```
RETF    2                    ;Remove the original status flags on return (remember we got here via an INT)
```

```
;----- Floppy Disk Routines -----
```

```

FDISK_RESET:                                ;Home the disk head etc.

    PUSH    BX                                ;Save ALL
    PUSH    CX
    PUSH    DX
    MOV     [CURRENT_DRIVE],DL

    MOV     CL,CMD_SET_DRIVE                 ;Set Drive Drive, ZFDC will just return if current drive
    CALL    S100OUT
    MOV     CL,[CURRENT_DRIVE]
    OR      CL,CL                            ;DL = 0 --> ZFDC Drive #3.  DL = 1 -->ZFDC Drive #2
    MOV     CL,3                             ;Default to Drive #3
    JZ      R_FLOPPY
    MOV     CL,2                             ;Drive #2

R_FLOPPY:
    CALL    S100OUT
    CALL    WAIT_FOR_ACK                     ;Return Z (or NZ with error # in [AH])
    JNZ     FDISK_RESET_ERROR

    MOV     CL,CMD_SET_HOME                  ;Home the heads of the current drive
    CALL    S100OUT
    CALL    WAIT_FOR_ACK                     ;Return Z (or NZ with error # in [AH])
    JNZ     FDISK_RESET_ERROR

    mov     byte [SEEK_STATUS],0            ;show good seek status
    mov     byte [IBM_DISK_STATUS],0        ;and good disk status

```

```

POP    DX

POP    CX

POP    BX

JMP    DONE_DISK          ;and return

```

FDISK_RESET_ERROR:

```

MOV     BX,HOME_ERR_MSG

CALL    PRINT_STRING

mov     byte [IBM_DISK_STATUS],seekerr ;show seek error

mov     byte [VERIFY_FLAG],0           ;Initially sector reads (rather than sctor verifys)

POP     DX

POP     CX

POP     BX

JMP     DONE_DISK          ;and return (with error)

```

FDISK_STATUS: ;AH = 0

```

mov     al,[IBM_DISK_STATUS]           ;Return past floppy status

mov     byte [IBM_DISK_STATUS],0 ;reset status in low RAM for next time

JMP     DONE_DISK          ;and return

```

FDISK_PARAMS: ;AH = 8

```

CMP     DL,00H                ;DL=1 from INT call if drive B: (ZFDC controller drive #2, 5" disk)
JZ      IS_144M_DISK
MOV     DI,FDISK_5PARM_TBL    ;Return with drive paramater table in ES:DI
MOV     AX,CS
MOV     ES,AX                ;And segment int ES:
XOR     AX,AX                ;Disk paramaters for 360K 5" Drive
MOV     BH,0                 ;Always
MOV     BL,01H               ;0=Unknown, 1=360K, 2=1.2M, 3=720K, 4=1.44M
MOV     CH,27H                ;Max Track 39
MOV     CL,9                  ;Max sector
MOV     DH,1                 ;Max value of head
MOV     DL,2                 ;Number of floppy disks
mov     byte [IBM_DISK_STATUS],0 ;Show OK
JMP     DONE_DISK            ;and return

```

IS_144M_DISK:

```

MOV     DI,FDISK_3PARM_TBL    ;Return with drive paramater table in ES:DI
MOV     AX,CS
MOV     ES,AX                ;And segment int ES:
XOR     AX,AX                ;Disk paramaters for 1.44M 3" Drive
MOV     BH,0                 ;Always
MOV     BL,04H               ;0=Unknown, 1=360K, 2=1.2M, 3=720K, 4=1.44M
MOV     CH,4FH                ;Max Track 79
MOV     CL,18                 ;Max sector
MOV     DH,1                 ;Max value of head

```

```

MOV     DL,2                      ;Number of floppy disks!
mov     byte [IBM_DISK_STATUS],0 ;Show OK
        JMP  DONE_DISK           ;and return

```

```

FDISK_DASB:                      ;AH = 15H, GET DISK TYPE (XT 1986/1/10 or later,XT286,AT,PS)
        XOR  AX,AX
        MOV  AH,01                ;For now return flag "no disk change line support implemmented"
        mov  byte [IBM_DISK_STATUS],0 ;Show OK
        CLC                      ;Clear CF
        JMP  ALL_OK              ;Do not check status, just return

```

```

FDISK_MEDIA_CHANGE:             ;AH = 16H
        XOR  AX,AX
        MOV  AX,06                ;change line not support implemmented"
        mov  byte [IBM_DISK_STATUS],0 ;Show OK
        CLC                      ;Clear CF
        JMP  ALL_OK              ;Do not check status, just return

```

```

;----- READ FLOPPY DISK SECTORS -----

```

```

FDISK_READ:                     ;AH=2, Read disk sector(s)

```

```

PUSH  BX                      ;Save everything, DS already on stack
PUSH  CX
PUSH  DX
PUSH  ES
PUSH  DI                      ;Used in LES below and DMA_ADJUST

MOV   [SECTORS_TO_DO],AL      ;save everything first
MOV   byte [SECTORS_DONE],0
MOV   [CURRENT_TRACK],CH
MOV   [CURRENT_SECTOR],CL
MOV   [CURRENT_HEAD],DH
MOV   [CURRENT_DRIVE],DL
MOV   [DMA_SEGMENT],ES       ;Save for below
MOV   [DMA_OFFSET],BX

CALL  DMA_ADJUST             ;Some DMA controllers cannot cross seg boundries, adjust

READ_COMMON:
MOV   CL,CMD_SET_DRIVE       ;Set Drive Drive, ZFDC will just return if current drive
CALL  S100OUT
MOV   CL,[CURRENT_DRIVE]     ;DL from INT call
OR    CL,CL                  ;DL = 0 --> ZFDC Drive #3.  DL = 1 -->ZFDC Drive #2
MOV   CL,3                   ;Default to Drive #3
JZ    RDD_FLOPPY
MOV   CL,2                   ;Drive #2

```


RDD_FLOPPY:

```

CALL    S100OUT

CALL    WAIT_FOR_ACK            ;Return Z (or NZ with error # in [AH])

JZ      READ_1

mov     byte [IBM_DISK_STATUS],seekerr ;show seek error

JMP     ZFDC_READ_ERROR

```

READ_1:

```

MOV     CL,CMD_SET_TRACK        ;Set Track

CALL    S100OUT

MOV     CL,[CURRENT_TRACK]

CALL    S100OUT                ;Send Selected track number

CALL    WAIT_FOR_ACK            ;Return Z (or NZ with error # in [AH])

JZ      READ_2

mov     byte [IBM_DISK_STATUS],seekerr ;show seek error

JMP     ZFDC_READ_ERROR

```

READ_2:

```

MOV     CL,CMD_SET_SIDE         ;Set Drive Side/Head

CALL    S100OUT

MOV     CL,[CURRENT_HEAD]       ;Set side (Head 0,1)

CALL    S100OUT

CALL    WAIT_FOR_ACK            ;Return Z (or NZ with error # in [AH])

JZ      READ_3

mov     byte [IBM_DISK_STATUS],seekerr ;show seek error

JMP     ZFDC_READ_ERROR

```

READ_3:

```

MOV    CL,CMD_DOS_SET_SECTOR          ;Set MS_DOS Sector (Note not CMD_SET_SECTOR for CPM)
CALL   S100OUT
MOV    CL,[CURRENT_SECTOR]
CALL   S100OUT                        ;Send Selected track number
CALL   WAIT_FOR_ACK                   ;Return Z (or NZ with error # in [AH])
JZ     READ_4
mov     byte [IBM_DISK_STATUS],seekerr ;show seek error
JMP     ZFDC_READ_ERROR

```

READ_4: MOV CL,CMD_SEEEK_TRACK ;Seek to that track (if not already there)

```

CALL   S100OUT
CALL   WAIT_FOR_ACK                   ;Return Z (or NZ with error # in [AH])
JZ     READ_5
mov     byte [IBM_DISK_STATUS],seekerr ;show seek error
JMP     ZFDC_READ_ERROR

```

READ_5: MOV CL,CMD_DOS_RD_MULTI_SEC ;Routine assumes required Drive Table,Drive,(Side),Track, and sector are already sent to board

```

CALL   S100OUT
MOV     CL,[SECTORS_TO_DO]             ;Count of sectors to be done (IBM says it will be (1---9 Max)
CMP     CL,18                          ;IBM says it will always be <= 18 for 3" (9 for 5")
JLE     READ_5a
mov     byte [IBM_DISK_STATUS],dmaerr  ;show as DMA error
JMP     ZFDC_READ_ERROR

```



```

RDSEC2:      IN      AL,S100STATUSB          ;Send data to ZFDC output

      TEST  AL,80H                          ;Is ZFDC in INPUT mode, if not wait
      JZ    RDSEC1

      TEST  AL,01H                          ;Has previous (if any) character been read.
      JZ    RDSEC1                          ;Z if not yet ready


      IN    AL,S100DATAB                    ;Get data
      STOSB                                ;READ 1 BYTE BYTE, AL->ES:[DI++]
      LOOP  RDSEC


RDSEC5:      mov     al,[SECTORS_DONE]        ;We have done one sector, are there more
      INC     al
      mov     [SECTORS_DONE],al             ;Store it
      CMP     [SECTORS_TO_DO],al            ;Have we done all yet
      JNZ     RD_LOOP

      CALL    WAIT_FOR_ACK                  ;Return Z (or NZ with error # in [AH])
      JNZ     RD_SEC_ERR


      mov     byte [IBM_DISK_STATUS],0      ;Show good operation
      POP     DI                            ;Get back all original registers
      POP     ES
      POP     DX
      POP     CX

```

```

POP     BX

mov     AL, [SECTORS_DONE]      ;Return # of sectors done

JMP     DONE_DISK              ;and return

```

RD_SEC_ERR:

```

mov     byte [IBM_DISK_STATUS],crcerr    ;Show CRC error

                                           ;Fall through to ZFDC_READ_ERROR

```

ZFDC_READ_ERROR: ;General read sector error reporting routine

```

PUSH    AX

MOV     BX, READ_ERR_MSG

CALL    PRINT_STRING

POP     AX

MOV     AL, AH

CALL    AL_HEXOUT

MOV     BX, H_MSG_CRLF

CALL    PRINT_STRING

POP     DI                        ;Get back all original registers

POP     ES

POP     DX

POP     CX

POP     BX

mov     AL, [SECTORS_DONE]      ;Return # of sectors done

JMP     DONE_DISK              ;and return

```

VERIFY_SECTOR: ;Special case where we just check sector for CRC errors/verify

```

        MOV     BX,SECTOR_TIMEOUT           ;Put in a timeout count (Loop for status reads at most 256X4 times)
VRDSEC1:DEC     BX                           ;Dec BC
        JNZ     VRDSEC2                     ;Will wait 400H times before timing out
        MOV     AH,TIMEOUT_ERROR            ;Send Timeout error
        mov     byte [IBM_DISK_STATUS],timerr ;show as timeout error
        JMP     ZFDC_READ_ERROR

VRDSEC2:IN      AL,S100STATUSB              ;Send data to ZFDC output
        TEST    AL,80H                     ;Is ZFDC in INPUT mode, if not wait
        JZ      VRDSEC1
        TEST    AL,01H                     ;Has previous (if any) character been read.
        JZ      VRDSEC1                   ;Z if not yet ready

        IN      AL,S100DATAB               ;Get data
        LOOP    VERIFY_SECTOR
        JMP     RDSEC5                     ;Are there more sectors to verify

```

```

;----- WRITE FLOPPY DISK SECTORS -----

```

```

FDISK_WRITE:           ;AH=3, Write disk
        PUSH    BX           ;Save everything, DS already on stack
        PUSH    CX
        PUSH    DX

```

```

PUSH    ES

PUSH    DI                ;Used in DMA_ADJUST

PUSH    SI                ;Need for LDS below


MOV     [SECTORS_TO_DO],AL    ;save everything first
MOV     byte [SECTORS_DONE],0
MOV     [CURRENT_TRACK],CH
MOV     [CURRENT_SECTOR],CL
MOV     [CURRENT_HEAD],DH
MOV     [CURRENT_DRIVE],DL
MOV     [DMA_SEGMENT],ES      ;Save for below
MOV     [DMA_OFFSET],BX


CALL    DMA_ADJUST          ;Some DMA controllers cannot cross seg boundries, adjust

```

WRITE_COMMON:

```

MOV     CL,CMD_SET_DRIVE    ;Set Drive Drive, ZFDC will just return if current drive
CALL    S100OUT

MOV     CL,[CURRENT_DRIVE]  ;DL from INT call
OR      CL,CL               ;DL = 0 --> ZFDC Drive #3.  DL = 1 -->ZFDC Drive #2
MOV     CL,3                ;Default to Drive #3
JZ      WDD_FLOPPY

MOV     CL,2                ;Drive #2

```

WDD_FLOPPY:

```

CALL    S100OUT

```

```

CALL    WAIT_FOR_ACK                ;Return Z (or NZ with error # in [AH])

JZ      WRITE_1

mov     byte [IBM_DISK_STATUS],seekerr ;Show seek error

JMP     ZFDC_WRITE_ERROR

WRITE_1:

MOV     CL,CMD_SET_TRACK            ;<<< Set Track

CALL    S100OUT

MOV     CL,[CURRENT_TRACK]

CALL    S100OUT                    ;Send Selected track number

CALL    WAIT_FOR_ACK                ;Return Z (or NZ with error # in [AH])

JZ      WRITE_2

mov     byte [IBM_DISK_STATUS],seekerr ;show seek error

JMP     ZFDC_WRITE_ERROR

WRITE_2:

MOV     CL,CMD_SET_SIDE              ;<<< Set Drive Side/Head

CALL    S100OUT

MOV     CL,[CURRENT_HEAD]           ;Set side (Head 0,1)

CALL    S100OUT

CALL    WAIT_FOR_ACK                ;Return Z (or NZ with error # in [AH])

JZ      WRITE_3

mov     byte [IBM_DISK_STATUS],seekerr ;show seek error

JMP     ZFDC_WRITE_ERROR

WRITE_3:

MOV     CL,CMD_DOS_SET_SECTOR        ;Set MS_DOS Sector (Note not CMD_SET_SECTOR for CPM)

```



```

CALL    S100OUT
MOV     CL,[CURRENT_SECTOR]
CALL    S100OUT                ;Send Selected track number
CALL    WAIT_FOR_ACK           ;Return Z (or NZ with error # in [AH])
JZ      WRITE_4
mov     byte [IBM_DISK_STATUS],seekerr ;show seek error
JMP     ZFDC_WRITE_ERROR

```

WRITE_4:

```

MOV     CL,CMD_SEEEK_TRACK      ;<<< Seek to that track (if not already there)
CALL    S100OUT
CALL    WAIT_FOR_ACK           ;Return Z (or NZ with error # in [AH])
JZ      WRITE_5
mov     byte [IBM_DISK_STATUS],seekerr ;show seek error
JMP     ZFDC_WRITE_ERROR

```

WRITE_5:MOV CL,CMD_DOS_WR_MULTI_SEC ;Routine assumes required Drive Table,Drive,(Side),Track, and sector are already sent to board

```

CALL    S100OUT
MOV     CL,[SECTORS_TO_DO]      ;Count of sectors to be done (IBM says it will be (1---9 Max)
CMP     CL,18                   ;IBM says it will always be <= 18 for 3" (9 for 5")
JLE     WRITE_5a
mov     byte [IBM_DISK_STATUS],dmaerr ;show as DMA error
JMP     ZFDC_WRITE_ERROR

```

WRITE_5a:CALL S100OUT ;Send sector count


```

LODSB                ;WRITE 1 BYTE BYTE, DS:[SI++] -> AL
OUT    S100DATAB,AL    ;Send it
LOOP    WRSEC

mov     al,[ES:SECTORS_DONE]    ;We have done one sector, are there more
INC     al
mov     [ES:SECTORS_DONE],al    ;Store it
CMP     [ES:SECTORS_TO_DO],al   ;Have we done all yet
JNZ     WR_LOOP               ;read next 512 bytes

POP     DS                ;Balance up stack
CALL    WAIT_FOR_ACK        ;Return Z (or NZ with error # in [AH])
JNZ     WR_SEC_ERR

mov     byte [IBM_DISK_STATUS],0 ;Show good operation
POP     SI
POP     DI                ;Get back all origional registers
POP     ES
POP     DX
POP     CX
POP     BX

mov     AL, [SECTORS_DONE]    ;Return # of sectors done
JMP     DONE_DISK            ;and return

```

WR_SEC_ERR:

```
    mov     byte [IBM_DISK_STATUS],crcerr    ;Show CRC error

                                                ;Fall through to ZFDC_WRITE_ERROR
```

ZFDC_WRITE_ERROR: ;General write sector error reporting routine

```
    CMP     AH,DISK_WP_ERR                ;Special case for Write Protected Disk error
    JZ      F_DISK_WP_ERROR

    PUSH    AX

    MOV     BX,WRITE_ERR_MSG

    CALL    PRINT_STRING

    POP     AX

    MOV     AL,AH

    CALL    AL_HEXOUT

    MOV     BX,H_MSG_CRLF

    CALL    PRINT_STRING
```

WP_DONE:POP SI

```
    POP     DI                ;Get back all original registers

    POP     ES

    POP     DX

    POP     CX

    POP     BX

    mov     AL, [SECTORS_DONE]    ;Return # of sectors done

    JMP     DONE_DISK            ;and return
```

F_DISK_WP_ERROR:

```
    mov     byte [IBM_DISK_STATUS],wpterr    ;Write protected disk
```

```
JMP  WP_DONE
```

```
;----- FORMAT FLOPPY DISK -----
```

```
;Code not tested/complete yet!
```

```
FDISK_FORMAT:      ;Format the current disk using the ZFDC format track command
```

```
    PUSH    BX
```

```
    PUSH    CX
```

```
    PUSH    DX
```

```
    MOV     [CURRENT_DRIVE],DL
```

```
    MOV     [CURRENT_TRACK],CH
```

```
    CMP     DL,0      ;If first track home heads
```

```
    JNZ     FORMAT_F1
```

```
    MOV     CL,CMD_SET_HOME      ;Note this is a restore with NO verify. (disk my not be formatted)
```

```
    CALL    S100OUT
```

```
    CALL    WAIT_FOR_ACK      ;Return Z (or NZ with error # in [AH])
```

```
    JZ      FORMAT_F1
```

```
    mov     byte [IBM_DISK_STATUS],seekerr ;show seek error
```

```
    JMP     ZFDC_FORMAT_ERROR
```

```
FORMAT_F1:
```

```
    MOV     CL,CMD_FORMAT_TRACK      ;Format a complete track on ZFDC controller
```

```
    CALL    S100OUT
```

```

MOV     CL,[CURRENT_TRACK]           ;Send the track number
CALL    S100OUT

MOV     CL,CONFIRM_FORMAT            ;Now send SPECIAL OK to FORMAT Disk flag
CALL    S100OUT

                                           ;<<< Now wait until track is formatted >>>
WAIT_F:  CALL    S100STAT              ;Wait until ZFDC Board is ready
        JNZ     TRACK_DONE            ;NZ, something there!
MOV     AH,1
int      16H                          ;KEYBOARD - CHECK FOR KEYSTROKE
JZ       WAIT_F                      ;Nothing, then wait some more
MOV     AH,0                          ;Get character
int      16H
CMP     AL,ESC                        ;Was an ESC character eneterd
JZ       ZFDC_FORMAT_ERROR
JMP     WAIT_F

TRACK_DONE:
CALL    S100IN                       ;Get returned Error # (Note this releases the SEND_DATA routine on the ZFDC board)
CMP     AL,NO_ERRORS_FLAG            ;Was SEND_OK/NO_ERRORS_FLAG sent back from ZFDC Board
JNZ     ZFDC_FORMAT_ERROR
        mov     byte [IBM_DISK_STATUS],0      ;and good disk status
POP     DX
POP     CX

```

```

POP    BX

JMP     DONE_DISK           ;and return

```

ZFDC_FORMAT_ERROR:

```

mov     byte [IBM_DISK_STATUS],cmderr    ;Show as CMD error

PUSH    AX

MOV     BX,FORMAT_ERR_MSG

CALL    PRINT_STRING

POP     AX

MOV     AL,AH

CALL    AL_HEXOUT

MOV     BX,H_MSG_CRLF

CALL    PRINT_STRING

POP     DX

POP     CX

POP     BX

mov     AL, [SECTORS_DONE]              ;Return # of sectors done

JMP     DONE_DISK                      ;and return

```

;----- HARD DISK Routines -----

;We will use for MS-DOS Drive C: the second IDE Drive.

;Leaving the first for CPM86 (or, later the second MS-DOS hard disk)

HD_COMMANDS:

TEST AH,AH ;Is it a FDisk reset

JNZ N_HDISK_RESET

JMP HDISK_RESET

N_HDISK_RESET:

CMP AH,1 ;Is it a HDISK status request

JNZ N_HDISK_STATUS

JMP HDISK_STATUS

N_HDISK_STATUS:

CMP AH,2 ;Is it a HDISK read request

JNZ N_HDISK_READ

MOV byte [VERIFY_FLAG],0H ;Turn off verify flag

JMP HDISK_READ

N_HDISK_READ:

CMP AH,3 ;Is it a HDISK write request

JNZ N_HDISK_WRITE

JMP HDISK_WRITE

N_HDISK_WRITE:

CMP AH,4 ;Is it a HDISK Verify request

JNZ N_HDISK_VERIFY

MOV byte [VERIFY_FLAG],0FFH ;Turn on verify flag


```

        JMP     HDISK_READ           ;Modified read

N_HDISK_VERIFY:

        CMP     AH,5                ;Is it a HDisk format request

        JNZ     N_HDISK_FORMAT

        JMP     HDISK_FORMAT

N_HDISK_FORMAT:

        CMP     AH,8                ;Is it a HDisk paramaters request

        JNZ     N_HDISK_PARAMS

        JMP     HDISK_PARAMS

N_HDISK_PARAMS:

        CMP     AH,9                ;Is it a HDisk Controller Initilize request

        JNZ     N_HDISK_INIT_REQ

        JMP     HDISK_INIT_REQ

N_HDISK_INIT_REQ:

        CMP     AH,0DH              ;Is it a HDisk Reset request (Alternative)

        JNZ     N_HDISK_RESET2

        JMP     HDISK_RESET

N_HDISK_RESET2:

        CMP     AH,10H              ;Is it a HDisk Ready check request

        JNZ     N_HDISK_READY_CHK

        JMP     HDISK_READY_CHK

N_HDISK_READY_CHK:

        CMP     AH,15H              ;Is it a HDISK read DASB request

        JNZ     N_NOT_VALID_DISK

        JMP     HDISK_DASB

```

N_NOT_VALID_DISK:

```
JMP    NOT_VALID_DISK           ;Go to common/floppy error return
```

HDISK_RESET: ;AH = 0, Home the disk head etc.

```
CALL   SET_DRIVE_B              ;Select the second Drive/CF card as MS-DOS Drive C:
```

```
CALL   IDEinit                  ;Initialize drive 2. If there is no drive abort
```

```
JNZ    HDISK_RESET_ERROR
```

```
mov     byte [SEEK_STATUS],0      ;show good seek status
```

```
mov     byte [IBM_DISK_STATUS],0  ;and good disk status
```

```
JMP     DONE_DISK               ;and return
```

HDISK_RESET_ERROR:

```
MOV     BX,HOME_ERR_MSG
```

```
CALL    PRINT_STRING
```

```
mov     byte [IBM_DISK_STATUS],seekerr ;show seek error
```

```
JMP     DONE_DISK               ;and return (with error)
```

HDISK_STATUS: ;AH = 1

```
mov     al,[IBM_DISK_STATUS]      ;Return past disk status
```

```
mov     byte [IBM_DISK_STATUS],0  ;reset status in low RAM for next time
```

```
JMP     DONE_DISK               ;and return
```

```

HDISK_PARAMS:                ;AH = 8H Get Hard Drive Parm's (We will assume one hard disk only, Custom type)

    MOV     AH,0

    MOV     AL,DOS_MAXSEC      ;Do NOT change ES or BX

    MOV     CH,DOS_MAXCYL_L-1  ;0FEH, low eight bits of maximum cylinder number

    MOV     CL,DOS_MAXSEC_CYL  ;3FH, maximum sector number (bits 5-0)+ two Cyl High Bits (Sectors numbered 1....x)

                                ;high two bits of maximum cylinder number (bits 7-6)

    MOV     DH,DOS_MAXHEADS-1  ;0FH, (0...15) 16 Heads

    MOV     DL,1               ;Number of Hard Disks

    mov     byte [IBM_DISK_STATUS],0 ;Show OK

    JMP     DONE_DISK          ;and return. This will put AH=0 & Clear CF

```

```

HDISK_INIT_REQ:              ;AH = 9H,  INITIALIZE CONTROLLER WITH DRIVE PARAMETERS (AT,PS)

```

```

HDISK_READY_CHK:            ;AH = 10H,  HARD DISK - CHECK IF DRIVE READY

    mov     byte [IBM_DISK_STATUS],0 ;Since we have only one HDisk just return for now

    JMP     DONE_DISK

```

```

HDISK_DASB:                 ;AH = 15H,  GET DISK TYPE (XT 1986/1/10 or later,XT286,AT,PS)

    MOV     AX,0310H          ;AH, Indicates a Hard Disk

    MOV     CX,000FH

    MOV     DX,0BC10H         ;This is what the AMI Bios returns for our custom drive (CX:DX = Total sectors)

    mov     byte [IBM_DISK_STATUS],0 ;Show OK

    CLC                      ;Clear CF

    JMP     ALL_OK            ;Do not check status, just return

```

```

HDISK_FORMAT:                                ;AH = 05H,   Format disk - Return immediately with status ok

    mov     byte [IBM_DISK_STATUS],0 ;show good operation no matter what

    JMP     DONE_DISK                        ;and return

```

```

;----- READ HARD DISK DISK SECTORS -----

```

```

HDISK_READ:

    PUSH    BX                                ;Save everything, DS already on stack
    PUSH    CX
    PUSH    DX
    PUSH    ES
    PUSH    DI                                ;Used in LES below and DMA_ADJUST

    MOV     [SECTORS_TO_DO],AL                ;save everything first
    MOV     byte [SECTORS_DONE],0
    MOV     AL,CL                             ;Store Sector
    AND     CL,00111111B                      ;Strip High 2 track bits
    MOV     [CURRENT_SECTOR],CL
    MOV     AH,0                              ;Shift the top 2 bits of AL into AH
    SHL     AX,1
    SHL     AX,1
    MOV     [CURRENT_TRACK_HIGH],AH

```

```

MOV     [CURRENT_TRACK],CH           ;Store low track#
MOV     [CURRENT_HEAD],DH           ;Store Head#
MOV     [CURRENT_DRIVE],DL          ;Actully for now always drive #2 on IDE Board
MOV     [DMA_SEGMENT],ES            ;Save for below
MOV     [DMA_OFFSET],BX

CALL    DMA_ADJUST                   ;Some DMA controllers cannot cross seg boundries, adjust

CMP     byte [DEBUG_FLAG],2         ;Is Detailed Hdisk/Floppy Debug mode on
JL      HREAD_COMMON                ;If not skip
CALL    DUMP_TRACK_PARAMS            ;Dump the Track,Head,Cylinder data to serial debug terminal

```

HREAD_COMMON:

```

CALL    DOS_WR_LBA                   ;Setup Drive, Track, Sector for MS-DOS formatted disk.
CALL    IDEwaitnotbusy                ;Make sure drive is ready
JNB     HL_19                         ;Carry flag set if problem
CALL    SHOWerrors                    ;Show error data on CRT
mov     byte [IBM_DISK_STATUS],seekerr ;show seek error
JMP     H_READ_ERROR                 ;General read HDisk sector error reporting routine
                                           ;and return (with error)

```

```

HL_19: MOV     DH,COMMANDread
MOV     DL,REGcommand
CALL    IDEwr8D                       ;Send Sec read command to drive.
CALL    IDEwaitdrq                    ;Wait until it's got the data

```

```

JNB    HL_20                ;Carry flag set if problem

CALL   SHOWerrors           ;Show error data on CRT

mov     byte [IBM_DISK_STATUS],crcerr    ;Show as CRC error

JMP     H_READ_ERROR        ;General read HDisk sector error reporting routine
                                ;and return (with error)

HL_20: LES     DI,[DMA_OFFSET]          ;Point to initial DMA address (ES & DI already saved above)

HRD_LOOP:

MOV     CX,256               ;ALWAYS read 512 bytes to [CX] (256X2 bytes)

HRD_LOOP_BYTES:

MOV     AL,REGdata           ;REG regsiteer address

OUT     IDEportC,AL

OR      AL,IDERdline         ;08H+40H, Pulse RD line

OUT     IDEportC,AL

IN      AL,IDEportA          ;Read the LOWER byte first

STOSB                          ;READ 1 BYTE BYTE, AL->ES:[DI++]

IN      AL,IDEportB          ;THEN read the UPPER byte

STOSB                          ;READ 1 BYTE BYTE, AL->ES:[DI++]

MOV     AL,REGdata           ;Deassert RD line

OUT     IDEportC,AL

```

```

LOOP    HRD_LOOP_BYTES                ;256 words, for 512 bytes

```

```

CMP     byte [DEBUG_FLAG],2           ;Is Detailed HDISK/Floppy Debug mode on

```

```

JL      HRDSEC4                       ;If not skip

```

```

CALL    SERIAL_DUMP_RD_SECTOR_DATA ;Dump first 16 bytes of data

```

```

HRDSEC4:

```

```

MOV     CX,0FFFFH                    ;Need to wait until the IDE drive is ready

```

```

HRDSEC5:                               ;with status data after potentially a long

```

```

MOV     DL,REGstatus                  ;series of sector reads.

```

```

CALL    IDErd8D                      ;Returned data in DH

```

```

AND     DH,80H                       ;Is IDE Drive still busy (bit 7 low)

```

```

JZ      HRDSEC6                      ;No, then check returned status

```

```

LOOP    HRDSEC5                      ;wait until ready

```

```

HRDSEC6:

```

```

MOV     AL,DH                        ;Was previous command completed without errors

```

```

AND     AL,1H                        ;Ret AL=0 for all OK

```

```

JZ      HNEXT_SECTOR_RD

```

```

CALL    SHOWerrors                   ;Show error data on CRT

```

```

mov     byte [IBM_DISK_STATUS],crr    ;Show as CRC error

```

```

JMP     H_READ_ERROR                 ;General write HDisk sector error reporting routine
                                           ;and return (with error)

```

```

                                           ;We have done one sector, are there more

```

```

                                ;On hard disks (with XT and AT BIOSes), a multi-sector read
                                ;continues on the next higher head of the same cylinder and if
HNEXT_SECTOR_RD:                ;necessary, advances to the next higher cylinder on the first head.

    mov     al,[SECTORS_DONE]

    INC     al

    mov     [SECTORS_DONE],al    ;Store it

    CMP     [SECTORS_TO_DO],al   ;Have we done all yet

    JNE     HRD_LOOP

    mov     byte [IBM_DISK_STATUS],0 ;Show good operation

HRD_DONE:

    POP     DI                    ;Get back all original registers

    POP     ES

    POP     DX

    POP     CX

    POP     BX

    mov     AL, [SECTORS_DONE]    ;Return # of sectors done

    JMP     DONE_DISK            ;and return

H_READ_ERROR:

    PUSH    AX

    MOV     BX,HREAD_ERR_MSG

    CALL    PRINT_STRING

    CALL    H_PRINT_CHS           ;Print current Cyl, Head, Sector (DS: points to low RAM data stores)

    POP     AX

```



```
JMP     HRD_DONE
```

```
HDISK_WRITE:                ;Arrive here with DS: pointing to low RAM data area

    PUSH    BX                ;Save everything, DS already on stack
    PUSH    CX
    PUSH    DX
    PUSH    ES
    PUSH    DI                ;Used in LES below and DMA_ADJUST
    PUSH    SI                ;Need for LDS below

    MOV     [SECTORS_TO_DO],AL    ;save everything first
    MOV     byte [SECTORS_DONE],0
    MOV     AL,CL                ;Store Sector
    AND     CL,00111111B         ;Strip High 2 track bits
    MOV     [CURRENT_SECTOR],CL
    MOV     AH,0                ;Shift the top 2 bits of AL into AH
    SHL     AX,1
    SHL     AX,1
    MOV     [CURRENT_TRACK_HIGH],AH
    MOV     [CURRENT_TRACK],CH    ;Store low track#
    MOV     [CURRENT_HEAD],DH    ;Store Head#
    MOV     [CURRENT_DRIVE],DL   ;Actully for now always drive #2 on IDE Board
```

```

MOV     [DMA_SEGMENT],ES           ;Save for below
MOV     [DMA_OFFSET],BX

CALL    DMA_ADJUST                 ;Some DMA controllers cannot cross seg boundries, adjust

CMP     byte [DEBUG_FLAG],2       ;Is Detailed Hdisk/Floppy Debug mode on
JL      HWRITE_COMMON             ;If not skip
CALL    DUMP_TRACK_PARAMS         ;Dump the Track,Head,Cylinder data to serial debug terminal

```

HWRITE_COMMON:

```

CALL    DOS_WR_LBA                ;Setup Drive, Track, Sector for MS-DOS formatted disk.
CALL    IDEwaitnotbusy            ;Make sure drive is ready
JNB     HW_19                    ;Carry flag set if problem
CALL    SHOWerrors               ;Show error data on CRT
mov     byte [IBM_DISK_STATUS],seekerr ;show seek error
JMP     H_WRITE_ERROR            ;General write HDisk sector error reporting routine
                                           ;and return (with error)

```

```

HW_19: MOV     DH,COMMANDwrite
MOV     DL,REGcommand
CALL    IDEwr8D                  ;Send Sec write command to drive.
CALL    IDEwaitdrq              ;Wait until it's got the data
JNB     HW_20                    ;Carry flag set if problem
CALL    SHOWerrors               ;Show error data on CRT
mov     byte [IBM_DISK_STATUS],crcerr ;Show as CRC error

```

```

        JMP     H_WRITE_ERROR           ;General write HDisk sector error reporting routine
                                         ;and return (with error)

HW_20:  PUSH    DS                     ;Remember from now on, low RAM DS pointer is no longer valid
        MOV     AX,DS
        XOR     AX,AX
        MOV     ES,AX                 ;ES will now temporarily have the low RAM pointer
        LDS     SI,[DMA_OFFSET]

HWR_LOOP:
        MOV     AL,WRITEcfg8255        ;8255 to write mode
        OUT     IDECtrlPort,AL
        MOV     CX,256                 ;ALWAYS read 512 bytes to [CH] (256X2 bytes)

HWR_LOOP_BYTES:
        LODSB                          ;WRITE 1 BYTE, DS:[SI++] -> AL
        OUT     IDEportA,AL            ;Write the LOWER byte first
        LODSB                          ;WRITE 1 BYTE, DS:[SI++] -> AL
        OUT     IDEportB,AL            ;THEN UPPER byte on B

        MOV     AL,REGdata
        PUSH    AX
        OUT     IDEportC,AL            ;Send write command
        OR      AL,IDEwrlne            ;Send WR pulse

```

```

OUT    IDEportC,AL
POP    AX
OUT    IDEportC,AL          ;Send write command
LOOP   HWR_LOOP_BYTES      ;One sector done

MOV    AL,READcfg8255      ;Set 8255 back to read mode
OUT    IDECtrlPort,AL

MOV    CX,0FFFFH          ;Need to wait until the IDE drive is ready
HW_21:                      ;with status data after potentially a long
MOV    DL,REGstatus        ;Series of sector writes
CALL   IDErd8D             ;Returned data in DH
AND    DH,80H              ;Is IDE Drive still busy
JZ     HW_22                ;No, then check returned status
LOOP   HW_21

HW_22:
MOV    AL,DH               ;Was previous command completed without errors
AND    AL,1H               ;Ret AL=0 for all OK
JZ     HNEXT_SECTOR_WR

POP    DS                  ;Get back DS for above
CALL   SHOWerrors          ;Show error data on CRT
mov     byte [IBM_DISK_STATUS],crrerr ;Show as CRC error
JMP    H_WRITE_ERROR       ;General write HDisk sector error reporting routine

```

```

;We have done one sector, are there more
;On hard disks (with XT and AT BIOSes), a multi-sector read
;continues on the next higher head of the same cylinder and if
HNEXT_SECTOR_WR:      ;necessary, advances to the next higher cylinder on the first head.

    mov     al,[ES:SECTORS_DONE]
    INC     al
    mov     [ES:SECTORS_DONE],al      ;Store it
    CMP     [ES:SECTORS_TO_DO],al     ;Have we done all yet
    JNZ     HWR_LOOP

    POP     DS                        ;Get back DS for above.
    mov     byte [IBM_DISK_STATUS],0 ;Show good operation

HWR_DONE:
    POP     SI
    POP     DI                        ;Get back all original registers
    POP     ES
    POP     DX
    POP     CX
    POP     BX
    mov     AL,[SECTORS_DONE]         ;Return # of sectors done
    JMP     DONE_DISK                ;and return

H_WRITE_ERROR:
    MOV     AL,READcfg8255            ;Set 8255 back to read mode
    OUT     IDECtrlPort,AL

```

```

PUSH    AX

MOV     BX,HWRITE_ERR_MSG      ;"HDisk Sector Write Error"

CALL    PRINT_STRING

CALL    H_PRINT_CHS            ;Print current Cyl, Head, Sector (DS: points to low RAM data stores)

POP     AX

JMP     HWR_DONE

```

```

H_PRINT_CHS:                    ;DS: points to low RAM data stores

```

```

MOV     BX,HD_MSG              ;"  Head = "

CALL    PRINT_STRING

MOV     AL,[CURRENT_HEAD]

CALL    AL_HEXOUT

```

```

MOV     BX,CYL_MSG             ;"H  Cyl = "

CALL    PRINT_STRING

MOV     AH,[CURRENT_TRACK_HIGH]

MOV     AL,[CURRENT_TRACK]

CALL    AX_HEXOUT

```

```

MOV     BX,SEC_MSG             ;"H  Sec = "

CALL    PRINT_STRING

MOV     AL,[CURRENT_SECTOR]

CALL    AL_HEXOUT

```

```

MOV     BX,BRAC1_MSG           ;"H  ("

```

```

CALL    PRINT_STRING
MOV     AL,[SECTORS_DONE]
CALL    AL_HEXOUT

```

```

MOV     BX,OF_MSG           ;"H of "
CALL    PRINT_STRING
MOV     AL,[SECTORS_TO_DO]
CALL    AL_HEXOUT

```

```

MOV     BX,BRAC2_MSG        ;H") "
CALL    PRINT_STRING
RET

```

;----- SUPPORT ROUTINES FOR ZFDC BOARD FOR MSDOS/FREEDOS -----

```

INIT_ZFDC:                ;Return 0FFH in [ZFDC_INIT_FLAG] and Z flag set if all OK
    OUT     RESETZFDCPORT,AL    ;Do a hardware reset. Does not matter what is in [AL]

    MOV     AX,5                ;~0.5 second at 10 MHz
    MOV     CX,0                ;Delay to allow board to setup hardware
WAITD: LOOP    WAITD           ;Delay for ~0.5 seconds
    DEC     AX
    JNZ     WAITD

```

```

IN      AL,S100DATAB      ;Check the board is there

CMP     AL,CMD_HANDSHAKE  ;Make sure we get HANDSHAKE byte back

MOV     AH,ZFDC_ABSENT    ;If not then no ZFDC board present

JNZ     BADZFDC           ;If not there, just abort


MOV     AL,CMD_HANDSHAKE  ;Send another byte just to be sure.

OUT     S100DATAB,AL      ;This clears up ints on ZFDC board

CALL    WAIT_FOR_ACK      ;Return Z (or NZ with error # in [AH])


OR      AL,AL

MOV     AH,ZFDC_INIT_ERROR ;If not then no ZFDC board present

JNZ     BADZFDC           ;just abort


                                ;Leave drives 0,1 UNFORMATTED/UNINITIALIZED for now


MOV     CL,CMD_SET_FORMAT ;Send Set Disk Format to Drive CMD for drive #3 (1.44M 3" disk)

CALL    S100OUT

MOV     CL,3              ;Floppy Drive 3, (ZFDC Board expects a 0H, 1H, 2H or 3H)

CALL    S100OUT

MOV     CL,IBM144         ;1.4M (For MSDOS) DDDS, 18 X 512 Byte Sectors, 80 Tracks. (See ZFDC Board Code for more info)

CALL    S100OUT

CALL    WAIT_FOR_ACK      ;Return Z (or NZ with error # in [AH])

JNZ     BADZFDC


MOV     CL,CMD_SET_FORMAT ;Send Set Disk Format to Drive CMD for drive #2 (360K 5" disk)

```



```

CALL    S100OUT
MOV     CL,2                ;Floppy Drive 2, (ZFDC Board expects a 0H, 1H, 2H or 3H)
CALL    S100OUT
MOV     CL,MSDOS2           ;5", IBM PC, MSDOS 2.x, 512 byte, DDDS, 9 sector format (See ZFDC Board Code for more info)
CALL    S100OUT
CALL    WAIT_FOR_ACK        ;Return Z (or NZ with error # in [AH])
JNZ     BADZFDC

MOV     CL,CMD_SET_DRIVE    ;<<< Set Drive Drive DOS A: ZFDC will just return if current drive
CALL    S100OUT
MOV     CL,3                ;Set drive #3 as the current drive
CALL    S100OUT
CALL    WAIT_FOR_ACK        ;Return Z (or NZ with error # in [AH])
JNZ     BADZFDC             ;just abort

PUSH    BX                  ;Return BX unaltered
MOV     AL,0FFH             ;Flag to indicate ZFDC board is setup OK
MOV     [ZFDC_INIT_FLAG],AL ;Note DS is already set for ROM usage in low RAM (400H)
MOV     BX,ZFDC_OK_MSG      ;Announce success
CALL    PRINT_STRING
POP     BX
XOR     AL,AL
RET                                ;Return Z for all OK

```

BADZFDC:

```

PUSH    BX                ;Return BX unaltered

MOV     AL,0H             ;Flag to indicate ZFDC board is NOT OK

MOV     [ZFDC_INIT_FLAG],AL ;Note DS is already set for ROM usage in low RAM (400H)

MOV     BX,ZFDC_FAIL_MSG  ;Announce failure

CALL    PRINT_STRING

POP     BX

XOR     AL,AL

DEC     AL

RET                                ;Return NZ WITH ERROR IN AH

```

```

DUMP_TRACK_PARAMS:        ;Dump the Track,Head,Cylinder data to serial debug terminal

```

```

MOV     CL,CR

CALL    SERIAL_OUT

MOV     CL,LF

CALL    SERIAL_OUT

MOV     CL,'h'

CALL    SERIAL_OUT

MOV     AL,[CURRENT_HEAD]

CALL    SERIAL_AL_HEXOUT

MOV     CL,'.'

CALL    SERIAL_OUT

MOV     CL,'t'

```

```

CALL    SERIAL_OUT

MOV     AL,[CURRENT_TRACK] ;Note DS is already set for ROM usage in low RAM (400H)

CALL    SERIAL_AL_HEXOUT


MOV     CL, '.'
CALL    SERIAL_OUT
MOV     CL, 's'
CALL    SERIAL_OUT
MOV     AL,[CURRENT_SECTOR]
CALL    SERIAL_AL_HEXOUT


MOV     CL, ' '
CALL    SERIAL_OUT
MOV     CL, '#'
CALL    SERIAL_OUT
MOV     AL,[SECTORS_TO_DO]
CALL    SERIAL_AL_HEXOUT
MOV     CL, ','
CALL    SERIAL_OUT
MOV     AL,[SECTORS_DONE]
CALL    SERIAL_AL_HEXOUT
MOV     CL, ' '
CALL    SERIAL_OUT
MOV     CL, ' '

```

```
CALL  SERIAL_OUT
```

```
RET
```

```
SERIAL_DUMP_RD_SECTOR_DATA:           ;Note this is only for sector reads. ES: is invalid for Writes
```

```
    PUSH  AX                        ;Show first 8 bytes of sector data on serial output (for debugging)
```

```
    PUSH  BX
```

```
    PUSH  CX
```

```
    PUSH  DI
```

```
    MOV   CX,16                    ;Show first 16 characters
```

```
DUMPS1:    MOV   AL,[ES:DI]         ;DI will have the current address
```

```
    CALL  SERIAL_AL_HEXOUT
```

```
    INC   DI
```

```
    LOOP  DUMPS1
```

```
    MOV   BX,CR_TAB_MSG            ;CR to next line then tab in 18 spacs (for multisector reads)
```

```
    CALL  SERIAL_PRINT_STRING
```

```
    POP   DI
```

```
    POP   CX
```

```
    POP   BX
```

```
    POP   AX
```

```
RET
```

```
SIMPLE_SECTOR_DUMP:                 ;Dump first CX bytes of sector data at ES:BX on CRT
```

```
    PUSH  DS
```

```

PUSH    BX
PUSH    CX

PUSH    BX
PUSH    CX

XOR     AX,AX                ;Set DS to data area for ROM usage in low RAM @ 400H....)
MOV     DS,AX

MOV     BX,SEQAT500          ;"First [CX] bytes of loaded Sector (@ES:BX) Head ="
CALL    PRINT_STRING
MOV     AL,[CURRENT_HEAD]
CALL    AL_HEXOUT

MOV     BX,TRACK_MSG         ;"H  Track ="
CALL    PRINT_STRING
MOV     AL,[CURRENT_TRACK]
CALL    AL_HEXOUT

MOV     BX,SEC_MSG           ;"H  Sector ="
CALL    PRINT_STRING
MOV     AL,[CURRENT_SECTOR]
CALL    AL_HEXOUT
MOV     BX,START_DATA_MSG    ;"H  Start of Data =  CR,LF"

```

```
CALL PRINT_STRING
```

```
POP CX ;From above
```

```
POP BX
```

```
ONE_LINE_SECTOR1:
```

```
MOV AX,[ES:BX] ;High byte/low byte order
```

```
PUSH AX
```

```
CALL AL_HEXOUT
```

```
POP AX
```

```
MOV AL,AH
```

```
CALL AL_HEXOUT
```

```
INC BX
```

```
INC BX
```

```
LOOP ONE_LINE_SECTOR1
```

```
CALL CRLF
```

```
POP CX ;Again from above
```

```
POP BX
```

```
SECTOR_DUMP1:
```

```
PUSH CX
```

```
MOV CL,[ES:BX] ;High byte/low byte order
```

```
and cl,7fh
```

```
cmp cl,' ' ;filter out control characters
```

```
jnc xloop3
```

```

xloop4:      mov     cl, '.'
xloop3:      cmp     cl, '~'

             jnc     xloop4

             CALL    CO

             INC     BX                ;Next character

             POP     CX

             LOOP    SECTOR_DUMP1

             CALL    CRLF

             POP     DS                ;Balance up stack

             RET

S100STAT:                    ;Check if ZFDC has any data for S-100 system

             IN      AL,S100STATUSB

             TEST    AL,01H            ;Anything there ?

             JZ      S100ST1           ;Return 0 if nothing

             XOR     AL,AL

             DEC     AL                ;Return NZ, & 0FFH in AL if something there

S100ST1:RET

S100IN:      IN      AL,S100STATUSB    ;Check if ZFDC has any data for S-100 system

             TEST    AL,80H            ;Is ZFDC in input mode, if not, wait

             JZ      S100IN            ;If low then ZFDC board is still in input mode, wait

```

```

TEST  AL,01H
JZ     S100IN
IN     AL,S100DATAA      ;return with character in AL
RET

```

```

S100OUT:IN  AL,S100STATUSB      ;Send data to ZFDC output (arrive with character to be sent in C)

TEST  AL,80H      ;Is ZFDC in output mode, if not wait
JNZ   S100OUT
TEST  AL,02H      ;Has previous (if any) character been read.
JZ    S100OUT      ;Z if not yet ready
MOV   AL,CL
OUT   S100DATAB,AL
RET

```

```

WAIT_FOR_ACK:      ;Delay to wait for ZFDC to return data. There is a timeout of about 2 sec.

PUSH  BX      ;This can be increased if you are displaying debugging info on the ZFDC
PUSH  DX      ;HEX LED display.
MOV   BX,0
MOV   DL,STATUSDELAY      ;Timeout, (about 2 seconds)

XWAIT1:  IN     AL,S100STATUSB      ;Check if ZFDC has any data for S-100 system
TEST  AL,80H      ;Is ZFDC in input mode
JZ    XWAIT2      ;if low then ZFDC is still in input mode
CALL  S100STAT      ;Wait until ZFDC Board sends something

```



```

JZ      XWAIT2
CALL    S100IN          ;Get returned Error # (Note this releases the SENDDATA routine on the ZFDC board)
MOV     AH,AL            ;<<< Store Error Code (if any) in AH
CMP     AL,NO_ERRORS_FLAG ;Was SENDOK/NOERRORSFLAG sent back from ZFDC Board
POP     DX              ;Balance up stack
POP     BX
RET                                ;Return NZ if problem, Z if no problem

```

```

XWAIT2:  DEC     BH
JNZ     XWAIT1          ;Try for ~2 seconds
DEC     BH
DEC     BL
JNZ     XWAIT1
DEC     BH
DEC     BL
DEC     DL
JNZ     XWAIT1
XOR     AL,AL
DEC     AL
MOV     AH,3FH          ;Flag as local Time out error
POP     DX              ;Balance up stack
POP     BX
RET                                ;Return NZ flag set if timeout & 0FFH in [AL]
                                ;Error code in AH

```

```
; Adjust DMASEG:DMAOFF via [ES:DI] so that the in DI is the
; smallest possible. This process is called normalization.
; Registers:    Only ES and DI altered
```

```
DMA_ADJUST:
```

```
    MOV     ES,[DMA_SEGMENT]
    MOV     DI,[DMA_OFFSET]

    PUSH    AX
    PUSH    DI

    SHR     DI,1                ; Get paragraph to low 12 bits
    SHR     DI,1                ; Shift 0's in at hi 4 bits
    SHR     DI,1
    SHR     DI,1

    MOV     AX,ES                ; Get segment to Bx
    ADD     AX,DI                ; Add in segment skew
    MOV     ES,AX                ; Restore dma segment
    POP     DI                  ; Get back original offset
    AND     DI,0FH              ; Only need within paragraph

    MOV     [DMA_SEGMENT],ES
    MOV     [DMA_OFFSET],DI      ;<<< Later use LES (or for Sec Write LDS)
    POP     AX
```

RET

;*****

;

; Non Maskable Interrupt Handler (for IBM-PC is int #2, or 08H in RAM)

;

;*****

```
NMI_hnd:                                ;Non Maskable Interrupt Handler (Note uses current stack!)

    PUSHF                                ;Should not get here. If so send warning and continue

    PUSH  AX

    PUSH  BX

    PUSH  CX

    MOV   BX,NMI_MSG                     ;Announce we got an NMI Interrupt

    CALL  PRINT_STRING                   ;Note PRINT_STRING always uses the CS: override for the BX pointer

    POP   CX

    POP   BX

    POP   AX

    POPF                                  ;Note NMI does not push the flags on to the stack

    IRET
```

;*****

```

;
;      Print Screen Software Interrupt Handler (14H in RAM)
;
;*****
PrintScrTest:                ;Test interrupt from Monitor Menu
    MOV     BX,PSCR_TEST_MSG    ;Announce we are goint to print the screen
    CALL    PRINT_STRING        ;Note PRINT_STRING always uses the CS: override for the BX pointer

    MOV     AX,0001H            ;AH = 01h Initilize Printer
    MOV     DX,0                ;DX = printer number (00h-02h)
    INT     17H                ;Printer Int

    PUSH    ES                  ;Used for INT 10H Write String
    PUSH    BP
    PUSH    DS                  ;Need DS=0
    XOR     AX,AX               ;Set DS to data area for ROM usage in low RAM @ 400H....)
    MOV     DS,AX

    MOV     AX,word [CONSOL_FLAG] ;Save current Console Output device flag
    PUSH    AX
    MOV     word[CONSOL_FLAG],1 ;Set (temporly) Console Output device flag to CGA/VGA

    MOV     AX,CS
    MOV     ES,AX

    MOV     BP,PSCR_TEST_MSG    ;ES:BP = string pointer for INT 10H
    MOV     CX,PSCR_TEST_LEN    ;CX = string length

```

```

MOV    AH,13H                ;AH = 0EH, String Output mode
MOV    AL,1                  ;Update cursor, no attribute
MOV    BH,0                  ;Page 0
MOV    BL,02                 ;Attribute B/W
MOV    DX,0                  ;0,0
INT     10H

INT     5H                   ;Call the Print Screen Interrupt routine below

MOV    AX,000CH              ;AH = 00h PRINTER - WRITE CHARACTER, AL=0CH, Flush printer buffer
MOV    DX,0                  ;DX = printer number (00h-02h)
INT     17H                  ;Printer Int

POP     AX                   ;Get back saved current Console Output device flag
MOV     [CONSOL_FLAG],AX     ;DS still = 0
POP     DS
POP     BP
POP     ES
RET

```

```

PrintScreenRoutine:          ;In PC BIOS is at 0FF54H

    STI                      ;Can allow further interrupts

    push    ds
    push    ax

```

```

push    bx
push    cx
push    dx
xor     ax,ax
mov     ds,ax
cmp     byte[STATUS_BYTE],1 ;are we already here
jz      pexit
CMP     word [CONSOL_FLAG],0      ;Skip if current Console is Propeller board (0)
JZ      pexit

mov     Byte [STATUS_BYTE],byte 1
MOV     AH,15                    ;request current screen mode
INT     10H                      ;AL=mode, AH=columns,BH=page

mov     cl,ah
mov     ch,25                    ;CX= row & columns count
call    pcr1f

push    cx
mov     ah,3                    ;Get current cursor position

int     10H

pop     cx
push    dx                      ;Save cursor position to DX

```

```

        xor     dx,dx                ;to position 0,0

pri10:  mov     ah,2                  ;Directly from IBM ROM BIOS
        int     10h                  ;set cursor
        mov     ah,8
        int     10h                  ;read character
        or      al,al                ;valid character?
        jnz     pri15
        mov     al,' '
pri15:  push    dx
        xor     dx,dx
        xor     ah,ah
        int     17H                  ;print character
        pop     dx
        test    ah,25h               ;check for error
        jnz     err10
        inc     dl
        cmp     cl,dl
        jnz     pri10                ;next character
        xor     dl,dl                ;to col 0
        mov     ah,dl
        push    dx
        call    pcr1f
        pop     dx
        inc     dh

```

```

        cmp     ch,dh                ;all lines done?
        jnz     pri10

pri20: pop     dx                    ;Done, recall cursor position
        mov     ah,2
        int     10h
        mov     byte[STATUS_BYTE],0
        jmp     pexit

err10: pop     dx
        mov     ah,2
        int     10H                ;Restore cursor position

err20: mov     byte[STATUS_BYTE],0FFH ;Flag error
pexit: pop     dx
        pop     cx
        pop     bx
        pop     ax
        pop     ds
        IRET                        ;Note IRET

pcrlf: xor     dx,dx
        xor     ah,ah

```



```

mov    al,LF
int     17H
XOR     ah,ah
mov     al,CR
int     17H
ret

```

```

;*****

```

```

;

```

```

;      Keypressed Handler      (for IBM-PC is int #9 via 8259A to 24H in RAM)

```

```

;

```

```

;      IRQ1 - KEYBOARD DATA READY

```

```

;      This interrupt is generated when data is received from the keyboard. This is normally

```

```

;      a scan code (from either a keypress OR a key release), but may also be an ACK or NAK

```

```

;      of a command on AT-class keyboards. (Note My Propeller Board translates the scan codes ASCII chars)

```

```

;

```

```

;      Note: This IRQ may be masked by setting bit 1 on the 8259A I/O port 21h.

```

```

;

```

```

;      If the BIOS supports an enhanced (101/102-key)keyboard, it calls INT 15/AH=4Fh after reading the

```

```

;      scan code from the keyboard and before further processing all further processing uses the scan code

```

```

;      returned from INT 15/AH=4Fh. (This is not done here)

```

```

;

```

```

;      The default interrupt handler is at F000h:E987h in 100%-compatible BIOSes. The interrupt handler performs

```

```

;           the following actions for certain special keystrokes:-
;
;           Ctrl-Break clear keyboard buffer, place word 0000h in buffer, invokes INT 1B, and sets flag at 0040h:0071h
;           SysReq invokes INT 15/AH=85h (SysReq is often labeled SysRq)
;           Ctrl-Numlock place system in a tight wait loop until next INT 09
;           <<<<<< None of the above "extra" items are yet implemented >>>>>>
;
;           Shift-PrtSc invokes INT 05 (This is now implemented here).
;           Ctrl-Alt-Del jump to BIOS startup code (either F000h:FFF0h or the destination of the jump at that address)
;           Pause & Ctrl-Num_Lock will halt Console Output until the next keyboard input
;
;
;*****

```

```

keyhnd: push    ax                ;This interrupt can strike any time, so save all
      push     ds
      push     bx
      XOR      AX,AX              ;Set DS to data area for ROM usage in low RAM @ 400H....)
      MOV      DS,AX

      in       al,KEYIN           ;Get KEYBOARD DATA
      CMP      AL,1CH             ;Propeller Board will return 1CH for a "Print Screen" key press
      JNZ      NO_PRN_SCR
      mov      al,NS_EOI          ;End the current interrupt
      OUT      MASTER_PIC_PORT,al

```

```

INT     5H                ;Activate software Print Screen Interrupt

MOV     AX,000CH          ;AH = 00h PRINTER - WRITE CHARACTER, AL=0CH, Flush printer buffer
MOV     DX,0              ;DX = printer number (00h-02h)
INT     17H              ;Printer Int
POP     BX
JMP     K27               ;Finish up

```

NO_PRN_SCR:

```

CMP     AL,1DH            ;Propeller Board will return 1DH for "Ctrl+Alt+Del" key press
JNZ     NO_CAD_SCR
mov     al,NS_EOI         ;End the current interrupt
OUT     MASTER_PIC_PORT,al
JMP     word 0F000H:0FFF0H ;Far Jump to F000H:FFF0H ;To Reset the CPU address

```

NO_CAD_SCR:

```

CMP     AL,1EH            ;Propeller Board will return 1EH for "Pause" key press
JNZ     NO_PAUSE_SCR
OR      byte[KB_FLAG_1],HOLD_STATE ;Set flag to indicate a Pause is required
mov     al,NS_EOI         ;End the current interrupt
OUT     MASTER_PIC_PORT,al
POP     BX
pop     ds
pop     ax

```

IRET

NO_PAUSE_SCR:

AND byte[KB_FLAG_1],NO_HOLD_STATE ;Set flag to indicate a NO PAUSE is required

and al,7fh ;strip parity bit (if any)

mov bl,[chrcnt] ;get current character count

cmp bl,chrmax ;is the buffer full?

jge keyxt ;ignore if buffer full

inc bl

mov [chrcnt],bl ;store new character count

mov bx,[buftl] ;get destination address

mov [bx],al ;store the character

inc bx ;bump buffer address

cmp bx,keybuff+32 ;at end of buffer?

jlt keyhn1 ;skip if not

mov bx,keybuff ;reset to start of buffer

keyhn1: mov [buftl],bx ;store adr for next character

keyxt: pop bx

mov al,NS_EOI

OUT MASTER_PIC_PORT,al

K27: pop ds

pop ax

iret

```

;*****
;
;      Timer Handler          (for IBM-PC is int #8 via 8259A to 20H in RAM)
;      IRQ0 - SYSTEM TIMER
;
;      On a PC this is generated 18.2 times per second by channel 0 of the 8254 system timer.
;
;      This interrupt is used to keep the time-of-day clock updated. It can strike any time in a program!
;
;*****

;Note:  The IBM PC clock interrupts at
;1193180/65536 counts/sec (Approx 18.2 per second).
;Our clock interrupts at ~60 hz so adjust to approximate
;the IBM clock, the time constants in this routine must be
;adjusted accordingly if accurate time is to be kept by PC-DOS.

timer:  push ax                ;This interrupt can strike any time, so save all (flags are already saved)
        push  ds
        XOR   AX,AX           ;Set DS to data area for ROM usage in low RAM @ 400H....)
        MOV   DS,AX

        inc   word [timlow]    ;Bump count
        jnz   timer1

        inc   word [timhi]     ;Bump high part of count

timer1:  cmp    word [timhi],18h ;End of a day?

```



```

;*****

;

;      Time of Day Handler      (For IBM-PC Software Interrupt 1AH)

;

;Input AH = 00h      TIME - GET SYSTEM TIME

;Return:CX:DX = number of clock ticks since midnight

;      AL = midnight flag, nonzero if midnight passed since time last read

;

;

;Input AH = 01h      TIME - SET SYSTEM TIME

;      CX:DX = number of clock ticks since midnight

;Return:Nothing

;

;

;Input:      AH = 02h      TIME - GET REAL-TIME CLOCK TIME (AT, XT286, PS)

;      CF clear to avoid a bug

;Return:CF clear if successful

;      CH = hour (BCD)

;      CL = minutes (BCD)

;      DH = seconds (BCD)

;      DL = daylight savings flag (00h standard time, 01h daylight time)

```

```

;      CF set on error (i.e. clock not running or in middle of update)
;
;
;Input:      AH = 03h      TIME - SET REAL-TIME CLOCK TIME (AT, XT286, PS)
;      CH = hour (BCD)
;      CL = minutes (BCD)
;      DH = seconds (BCD)
;      DL = daylight savings flag (00h standard time, 01h daylight time)
;Return:Nothing
;
;
;Input:      AH = 04h      TIME - GET RTC DATE (AT, XT286, PS)
;      CH = century (BCD)
;      CL = year (BCD)
;      DH = month (BCD)
;      DL = day (BCD)
;      CF clear if OK
;
;
;Return:CF clear if successful
;      CH = century (BCD)
;      CL = year (BCD)
;      DH = month (BCD)
;      DL = day (BCD)
;      CF set on error

```



```

;
;
;Input:      AH = 05h      TIME - SET REAL-TIME CLOCK DATE (AT, XT286, PS)
;      CH = century (BCD)
;      CL = year  (BCD)
;      DH = month  (BCD)
;      DL = day   (BCD)
;Return:Nothing
;
;*****

```

```
time_of_day:
```

```

    sti
    push    ds
    PUSH    AX
    XOR     AX,AX                ;Set DS to data area for ROM usage in low RAM @ 400H....)
    MOV     DS,AX
    POP     AX

    CMP     byte [DEBUG_FLAG],0 ;Is Debug mode on
    JZ      Xtime_of_day
    CMP     AH,00H              ;Skip simple Get System Time
    JZ      Xtime_of_day
    PUSH    AX
    PUSH    BX

```

```

PUSH    CX

MOV     BX,INT_1AH_MSG           ;"Int 1AH (Time) AX="

CALL    SERIAL_PRINT_STRING

POP     CX

POP     BX

POP     AX

CALL    SERIAL_DISPLAY_REGISTERS ;Display Registers on serial port display (All registers retained)

```

Xtime_of_day:

```

TEST    AH,AH                   ;AH=0 read system tick time?

JZ      READ_TICKS              ;go do it if so

CMP     AH,1                    ;AH = 1 set tick time?

JZ      SET_TICKS

CMP     AH,2                    ;AH = 2 Get RTC Time?

JZ      READ_RTC_TIME

CMP     AH,4                    ;AH = 4 Get RTC Date?

JZ      READ_RTC_DATE

JMP     TIME_DONE

```

READ_TICKS: ;Read the system tick time

```

cli

mov     al,[timofl]

mov     byte [timofl],0

mov     cx,[timhi]

mov     dx,[timlow]

```

```

        sti

        pop     ds

        iret

SET_TICKS:                ;Set the system tick time

        cli                ;no interrupts while we set it

        mov     [timlow],dx

        mov     [timhi],cx

        mov     byte [timofl],0

        sti                ;interrupts ok now

TIME_DONE:

        pop     ds

        iret

                                ;AH = 2H, Read CMOS RTC Time

                                ;CH = hour (BCD, CL = minutes (BCD), DH = seconds (BCD), DL = daylight savings flag (00h standard
time, 01h daylight time)

READ_RTC_TIME:            ;CF set on error (i.e. clock not running or in middle of update)

        CALL    LOAD_TIME

        POP     ds                ;Get back the origional saved DS at start

        retf     2                ;Remove the origional status flags on return (remember we got here via an INT)

                                ;AH = 4H, Read CMOS RTC

                                ;CH = century (BCD), CL = year (BCD), DH = month (BCD), DL = day (BCD)

READ_RTC_DATE:            ;CF set on error

```

```

CALL    LOAD_DATE

POP     ds                ;Get back the original saved DS at start

retf    2                ;Remove the original status flags on return (remember we got here via an INT)

```

;----- Routines to set flag for Consol Output to Propeller or Lava Video board

SET_CO_FLAG:

```

MOV     BX,VIDIO_OUTPUT_MSG ;Video Board XY positioning etc tests. Enter AX Value

CALL    PRINT_STRING

CALL    CICO              ;1st Console input digit to AL

PUSH    DS

PUSH    BX

PUSH    AX

CALL    CRLF

POP     AX

XOR     BX,BX             ;Set DS to data area for ROM usage in low RAM @ 400H....)

MOV     DS,BX

CMP     AL,'0'

JZ      CRT_TO_PROPELLER

CMP     AL,'2'

JZ      CRT_TO_LAVA

CMP     AL,'1'

JZ      CRT_TO_VGA

```

```

MOV     BX,INT10_ERR_MSG      ;Invalid Selection

CMP     AL,ESC

JNZ     SET_CRT_DONE

POP     BX

POP     DS

RET

```

CRT_TO_PROPELLER:

```

MOV     word[CONSOL_FLAG],0 ;Console output Propeller Video Board

MOV     BX,VIDIO_PROP_MSG

CALL    SPEAK_STRING          ;Speak out the message

MOV     BX,VIDIO_PROP_SMSG    ;Video to PROPELLER

JMP     SET_CRT_DONE

```

CRT_TO_LAVA:

```

MOV     word[CONSOL_FLAG],2 ;Console output to LAVA Video Board

CALL    INITILIZE_LAVA        ;Clear Screen, Green/Black Cursor 0,0

MOV     BX,VIDIO_LAVA_SMSG

CALL    SPEAK_STRING          ;Speak out the message

MOV     BX,VIDIO_LAVA_MSG     ;Video to LAVA

JMP     SET_CRT_DONE

```

CRT_TO_VGA:

```

MOV     word[CONSOL_FLAG],1 ;Console output to CGA/VGA Video Board

MOV     BX,VIDIO_VGA_SMSG

CALL    SPEAK_STRING          ;Speak out the message

MOV     BX,VIDIO_VGA_MSG      ;Video to VGA

```

```
JMP    SET_CRT_DONE
```

```
SET_CRT_DONE:
```

```
CALL    PRINT_STRING
```

```
POP     BX
```

```
POP     DS
```

```
RET
```

```
;----- Routines to test Video Board Int 10H Functions out using this IBM PC BIOS section
```

```
;          The value of AH, CX, etc are used to control the positioning of characters on then CRT see below
```

```
XY_VIDEO:
```

```
MOV     BX,VIDIO_TEST_MSG    ;Video Board XY positioning etc tests. Enter AX Value
```

```
CALL    PRINT_STRING
```

```
CALL    GET4DIGITS
```

```
PUSH    DI                    ;AX value in DI
```

```
MOV     BX,ENTER_BX_MSG      ;Enter BX Value
```

```
CALL    PRINT_STRING
```

```
CALL    GET4DIGITS
```

```
PUSH    DI                    ;BX value in DI
```

```
MOV     BX,ENTER_CX_MSG      ;Enter CX Value
```

```
CALL    PRINT_STRING
```

```
CALL    GET4DIGITS
```

```
PUSH    DI                      ;CX value in DI
```

```
MOV     BX,ENTER_DX_MSG        ;Enter DX Value
```

```
CALL    PRINT_STRING
```

```
CALL    GET4DIGITS
```

```
PUSH    DI                      ;DX value in DI
```

```
MOV     BX,ACTIVATE_INT_MSG ;Will now activate the Int 10H command
```

```
CALL    PRINT_STRING
```

```
CALL    CICO
```

```
CMP     AL,ESC
```

```
JNZ     XY_VIDEO1
```

```
JMP     IBM_BIOS
```

```
XY_VIDEO1:                      ;Select below which hardware board you are testing
```

```
PUSH    DS
```

```
XOR     AX,AX
```

```
MOV     DS,AX
```

```
; MOV     word [CONSOL_FLAG],0      ;Send output to Propeller Video board
```

```
; MOV     word [CONSOL_FLAG],1      ;Send output to CGA/VGA Video board
```

```
MOV     word [CONSOL_FLAG],2      ;Send output to LAVA-10 Video board
```

```
POP     DS
```

```
POP     DX
```

```

POP     CX
POP     BX
POP     AX

INT     10H                ;Carry out the INT 10H interrupt

PUSH    DS
XOR     AX,AX
MOV     DS,AX
MOV     word [CONSOL_FLAG],0    ;Send output back to Propeller Board
POP     DS

MOV     CL,'# '
CALL    CO

CALL    CICO
CMP     AL,ESC
JNZ     XY_VIDEO
JMP     IBM_BIOS

```

```

;----- Routines to hold CPU in a loop so RAM read/write hardware signals can be analyzed

```

```

;Loop to analyze hardware RAM Read signals

```


READ_BYTE_TEST:

```

    MOV    BX,BYTE_RTEST_MSG    ;Get RAM location Enter AX Value
    CALL   PRINT_STRING
    CALL   GET5DIGITS            ;Will return ES=000xH, DI = xxxxH
    CALL   CRLF

```

```

BRTEST:    MOV    AL,[ES:DI]
           JMP    BRTEST

```

READ_WORD_TEST:

```

    MOV    BX,WORD_RTEST_MSG    ;Get RAM location Enter AX Value
    CALL   PRINT_STRING
    CALL   GET5DIGITS            ;Will return ES=000xH, DI = xxxxH
    CALL   CRLF

```

```

WRTEST:    MOV    AX,[ES:DI]
           JMP    WRTEST        ;Note CPU will not return from this loop. Reset required

```

;Loop to analyze hardware RAM Write signals

WRITE_BYTE_TEST:

```

    MOV    BX,BYTE_WTEST_MSG    ;Get RAM location Enter AX Value
    CALL   PRINT_STRING
    CALL   GET5DIGITS            ;Will return ES=000xH, DI = xxxxH
    CALL   CRLF
    MOV    AL,55H

```

```
BWTEST:      MOV     [ES:DI],AL
```

```
      JMP     BWTEST
```

```
WRITE_WORD_TEST:
```

```
      MOV     BX,WORD_WTEST_MSG    ;Get RAM location Enter AX Value
```

```
      CALL    PRINT_STRING
```

```
      CALL    GET5DIGITS           ;Will return ES=000xH, DI = xxxxH
```

```
      CALL    CRLF
```

```
      MOV     AX,5555H
```

```
WWTEST:      MOV     [ES:DI],AX
```

```
      JMP     WWTEST              ;Note CPU will not return from this loop. Reset required
```

```
;*****
```

```
;XXXXXX:
```

```
;      Video Output Handler      (Software Int# 10H)
```

```
;      Will recognize the following settings:-
```

```
;
```

```
;Input:      AH = 00h      VIDEO - SET VIDEO PARAMATERS
```

```
;      AL = Display Mode
```

```
;
```

```
;Input:      AH = 01h      VIDEO - SET TEXT-MODE CURSOR SHAPE
```

```
;      CH = cursor start and options (see below)
```

```
;      CL = bottom scan line containing cursor (bits 0-4)
```

```

;          Bitfields for cursor start and options:
;
;          7          should be zero
;
;          6,5       cursor blink.
;
;                  (00=normal, 01=invisible, 10=erratic, 11=slow).
;
;                  (00=normal, other=invisible on EGA/VGA)
;
;          4-0       topmost scan line containing cursor
;Return      Nothing
;
;Input:      AH = 02h      VIDEO - SET CURSOR POSITION
;
;          BH = page number (0-3 in modes 2&3. 0-7 in modes 0&1. 0 in graphics modes)
;
;          DH = row (00h is top)
;
;          DL = column (00h is left)
;Return:Nothing
;
;Input:      AH = 03h      VIDEO - GET CURSOR POSITION AND SIZE
;
;          BH = page number (0-3 in modes 2&3. 0-7 in modes 0&1. 0 in graphics modes)
;Return:AX = 0000h (Phoenix BIOS - only)
;
;          CH = start scan line of cursor
;
;          CL = end scan line of cursor
;
;          DH = row (00h is top)
;
;          DL = column (00h is left)
;
;Input:      AH = 05h      VIDEO - SET PAGE
;
;          BH = page number (0-3 in modes 2&3. 0-7 in modes 0&1. 0 in graphics modes)
;

```

```

;Input:      AH = 06h      VIDEO - SCROLL UP WINDOW

;      AL = number of lines by which to scroll up (00h = clear entire window)

;      BH = attribute used to write blank lines at bottom of window

;      CH,CL = row,column of window's upper left corner

;      DH,DL = row,column of window's lower right corner

;Return:Nothing

;

;Input:      AH = 07h      VIDEO - SCROLL DOWN WINDOW

;      AL = number of lines by which to scroll down (00h=clear entire window)

;      BH = attribute used to write blank lines at top of window

;      CH,CL = row,column of window's upper left corner

;      DH,DL = row,column of window's lower right corner

;Return:Nothing

;

;Input:AH = 08h      VIDEO - READ CHARACTER AND ATTRIBUTE AT CURSOR POSITION

;      BH = page number (00h to number of pages - 1) (see #00010)

;Return:      AH = character's attribute (text mode only)

;      AL = character

;

;Input:      AH = 09h      VIDEO - WRITE CHARACTER AND ATTRIBUTE AT CURSOR POSITION

;      AL = character to display

;      BH = page number (00h to number of pages - 1)

;      BL = attribute (text mode) or color (graphics mode)

;      if bit 7 set in <256-color graphics mode, character is XOR'ed onto screen

;      CX = number of times to write character

```

```
;Return:Nothing
```

```
;Input:      AH = 0Ah      VIDEO - WRITE CHARACTER ONLY AT CURSOR POSITION
```

```
;      AL = character to display
```

```
;      BH = page number (00h to number of pages - 1)
```

```
;      BL = attribute color (graphics mode)
```

```
;          if bit 7 set in <256-color graphics mode, character is XOR'ed onto screen
```

```
;      CX = number of times to write character
```

```
;Return:Nothing
```

```
;
```

```
;
```

```
;
```

```
;Input:      AH = 0Eh      VIDEO - TELETYPE OUTPUT
```

```
;      AL = character to write
```

```
;      BH = page number
```

```
;      BL = foreground color (graphics modes only)
```

```
;Return:Nothing
```

```
;
```

```
;
```

```
;Input:      AH = 0Fh      VIDEO - GET VIDEO PARAMATERS
```

```
;Return:
```

```
;      AH = Number of CRT Columns
```

```
;      AL = Display Mode
```

```
;      BH = Current page
```

```
;
```

```
;*****
```

```
VIDEO_TABLE:
```

```

DW    SET_MODE            ;<--0  Set Mode
DW    VIDEO_TBD           ;1     Set Cursor Type
DW    SET_CURSOR_POS      ;<--2  Set Cursor Positrion
DW    GET_CURSOR_POS      ;<--3  Get Cursor Position
DW    VIDEO_TBD           ;4     Read Light Pen
DW    SET_PAGE            ;<--5  Set page
DW    SCROOL_UP           ;<--6  Scoll up [AL] lines
DW    SCROOL_DOWN        ;<--7  Scroll down [AL] lines
DW    READ_CHAR_ATT       ;<--8  Read Char & Attribute at cursor position
DW    WRITE_AT_CURSOR_ATT ;<--9  Write character & attribute at current cursor position
DW    WRITE_AT_CURSOR     ;<--0AH  Write character at current cursor position
DW    VIDEO_TBD           ;0BH   Set Color
DW    VIDEO_TBD           ;0cH   Write Dot
DW    VIDEO_TBD           ;0dH   Read Dot
DW    VIDEO_TTY           ;<--0EH  ***** Simple TTY mode *****
DW    GET_VIDEO_PARMS     ;<--0FH  Get Video state
DW    VIDEO_TBD           ;10H   reserved
DW    VIDEO_TBD           ;11H   reserved
DW    VIDEO_VGA           ;12H   VIDEO - ALTERNATE FUNCTION SELECT (VGA, MCGA) - VIDEO ADDRESSING
DW    WRITE_STRING        ;13H   Write String

```

```
M1L    equ    $-VIDEO_TABLE
```

```

CONOUT: STI                ;For now just dump character on Propeller Console IO board

        CLD                ;This section of code will very carefully reproduce everything
                                ;that is in the IBM-PC BIOS ROM

        PUSH ES
        PUSH DS
        PUSH DX
        PUSH CX
        PUSH BX
        PUSH SI
        PUSH DI
        PUSH BP            ;New for AT BIOS
        PUSH AX            ;<<< Save character (in AH) on stack >>>
        PUSH AX            ;Note extra AX on stack for VIDEO_NOT_FINISHED etc routines below

                                ;Need for Debugging output below
        XOR  AX,AX          ;Set DS to data area for ROM usage in low RAM @ 400H....)
        MOV  DS,AX

        POP   AX

                                ;Save for AH=13H String write (Normally ES is used for the Video RAM pointer)
        MOV  [ES_STORE],ES

                                ;Skip simple TTYOut debugging (too much data)
        CMP  AH,0EH
        JZ   SKIP_VIDEO_DEBUG

        CMP  byte [DEBUG_FLAG],0 ;Is Debug mode on

```

```
JNZ    SHOW_DEBUG_DATA          ;If NZ, then debug mode
```

```
PUSH   AX
```

```
IN     AL,IOBYTE                ;If bit 3 of Port EFH is 0, Then force Debug Display
```

```
AND    AL,08H
```

```
POP    AX
```

```
JNZ    SKIP_VIDEO_DEBUG
```

```
SHOW_DEBUG_DATA:
```

```
PUSH   AX
```

```
PUSH   BX
```

```
MOV    BX,INT_10H_MSG           ;"Int 1AH (VIDEO) AX="
```

```
CALL   SERIAL_PRINT_STRING
```

```
POP    BX
```

```
POP    AX
```

```
CALL   SERIAL_DISPLAY_REGISTERS ;Display Registers on serial port display (All registers retained)
```

```
SKIP_VIDEO_DEBUG:              ;Use a lookup table to locate the correct AH option
```

```
MOV    AL,AH
```

```
XOR    AH,AH                    ;0 to AH
```

```
SAL    AX,1                    ;X2 for table lookup
```

```
MOV    SI,AX
```

```
CMP    AX,M1L                  ;Check we are within range
```

```
JB     VIDEO_AH_OK
```



```

PUSH  BX                      ;Out of range request
MOV   BX,INT_10H_MSG          ;"Int 1AH (VIDEO) AX="
CALL  SERIAL_PRINT_STRING
POP   BX
CALL  SERIAL_DISPLAY_REGISTERS ;Display Registers on serial port display (All registers retained)
POP   AX                      ;Currently there are 2 AX's on stack
JMP   VIDEO_RETURN

```

VIDEO_AH_OK:

```

MOV   AX,0B800H              ;Segment of CGA Board RAM
mov   di,[EQFLAG]
and   di,30h                 ;isolate crt switches
cmp   di,30h
jne   MX3
mov   ax,0B000h              ;segment for B/W card
MX3:  mov   es,ax             ;Set ES: to point to video area
POP   AX                     ;<--- Get requested AH & AL values
MOV   AH,[CRT_MODE]          ;Current mode now in AH (used by CGA/VGA video board)

JMP   [CS:SI+VIDEO_TABLE] ;go to appropriate routine with card type in DI
                                ;mode in AH, video ram in ES and value in AL.

```

;-----

```

SET_MODE:                      ;AH = 0h, AL= Mode VIDEO - GET VIDEO PRAMATERS

```

```

    CMP    word [CONSOL_FLAG],0        ;Send output to Propeller board (0), VGA Video board (1), or LAVA Board (2)
    JZ     PROPELLER_SET_MODE
    CMP    word [CONSOL_FLAG],2
    JZ     LAVA_SET_MODE
    JMP    VGA_SET_MODE                ;Set the display mode of the CGA/VGA Board

PROPELLER_SET_MODE:
    MOV    AX,0003H
    JMP    VIDEO_RETURN

LAVA_SET_MODE:                        ;LAVA has one mode only
    CALL   INITILIZE_LAVA              ;Clear Screen, Green/Black, Cursor 0,0
    MOV    AX,0003H
    JMP    VIDEO_RETURN

;-----

                                ;AH = 02h    VIDEO - SET CURSOR POSITION

SET_CURSOR_POS:                    ;DH = row (00h is top), DL = column (00h is left)
    CMP    word [CONSOL_FLAG],0        ;Send output to Propeller board (0),VGA Video board (1), or LAVA Board (2)
    JZ     PROPELLER_SET_CURSOR_POS
    CMP    word [CONSOL_FLAG],2
    JZ     LAVA_SET_CURSOR_POS
    JMP    VGA_SET_CURSOR_POS

PROPELLER_SET_CURSOR_POS:
    CALL   PROPELLER_SET_CURSOR        ;Set Cursor at [DX] on Propeller Board
    JMP    VIDEO_RETURN

LAVA_SET_CURSOR_POS:                ;Set Cursor at [DX] on LAVA Board

```

```

CALL    L_HIDE_CURSOR      ;DX is unchanged

CALL    L_SET_CURSOR

JMP     VIDEO_RETURN

```

```

;-----

```

```

;AH = 03h      VIDEO - GET CURSOR POSITION AND SIZE

```

```

GET_CURSOR_POS:                ;DH = row (00h is top), DL = column (00h is left)

```

```

    CMP     word [CONSOL_FLAG],0      ;Send output to Propeller board (0), or Video board (1), or LAVA Board (2)

    JZ      PROPELLER_GET_CURSOR_POS

    CMP     word [CONSOL_FLAG],2

    JZ      LAVA_GET_CURSOR_POS

    JMP     VGA_GET_CURSOR_POS

```

```

PROPELLER_GET_CURSOR_POS:

```

```

    JMP     VIDEO_NOT_FINISHED ;<<<< Ignore for now

```

```

LAVA_GET_CURSOR_POS:

```

```

    CALL    L_GET_CURSOR

    JMP     VIDEO_RETURN

```

```

;-----

```

```

;AH = 05h      VIDEO - SET PAGE

```

```

SET_PAGE:                ;BH = page number (0-3 in modes 2&3. 0-7 in modes 0&1. 0 in graphics modes)

```

```

    CMP     word [CONSOL_FLAG],0      ;Send output to Propeller board (0), or Video board (1), or LAVA Board (2)

    JZ      PROPELLER_SET_PAGE

```

```

    CMP    word [CONSOL_FLAG],2
    JZ     LAVA_SET_PAGE
    JMP    VGA_SET_PAGE

PROPELLER_SET_PAGE:
    MOV    BH,0                ;Always
    JMP    VIDEO_RETURN

LAVA_SET_PAGE:
    MOV    BH,0                ;<<<< Ignore for now (Add 0 & 1 later)
    JMP    VIDEO_RETURN

;-----

                ;AH = 06h    VIDEO - SCROLL UP WINDOW

SCROOL_UP:      ;AL = number of lines by which to scroll up (00h = clear entire window)
    CMP    word [CONSOL_FLAG],0    ;Send output to Propeller board (0), or Video board (1), or LAVA Board (2)
    JZ     PROPELLER_SCROOL_UP
    CMP    word [CONSOL_FLAG],2
    JZ     LAVA_SCROOL_UP
    JMP    VGA_SCROOL_UP

PROPELLER_SCROOL_UP:
    OR     CX,CX                ;Start 0,0?, (CH,CL = row,column start, DH,DL = row,column end)
    JZ     SCROLL_UP_0
    PUSH   AX
    PUSH   DX
    MOV    DX,CX                ;At least we will reposition cursor to 0,0

```

```

        CALL  PROPELLER_SET_CURSOR      ;Set Cursor at [DX]

        POP   DX

        POP   AX

        JMP   VIDEO_NOT_FINISHED  ;Will ignore DX for now

SCROLL_UP_0:

        OR    AL,AL                    ;AL has number of lines to scroll

        JNZ   SCROLL_UP_1

        MOV   AL,40                    ;0 for current 40 line CRT

SCROLL_UP_1:

        PUSH  AX

        MOV   AH,ESC

        CALL  FAST_CONOUT

        MOV   AH,'D'

        CALL  FAST_CONOUT

        POP   AX

        DEC   AL

        JNZ   SCROLL_UP_1

        JMP   VIDEO_RETURN

LAVA_SCROLL_UP:

        OR    CX,CX                    ;Start 0,0?, (CH,CL = row,column start, DH,DL = row,column end)

        JZ    LAVA_SCROLL_UP_L0

        PUSH  AX

        PUSH  DX

        MOV   DX,CX                    ;At least we will reposition cursor to 0,0

        CALL  L_SET_CURSOR             ;Set Cursor at [DX]

```

```

        POP     DX

        POP     AX

        JMP     VIDEO_NOT_FINISHED ;Will ignore DX for now

LAVA_SCROLL_UP_L0:

        OR      AL,AL                ;AL has number of lines to scroll

        JNZ     LAVA_SCROLL_UP_L1

        CALL    L_CLEAR_SCREEN      ;AL=0 then clear whole screen

        JMP     VIDEO_RETURN

LAVA_SCROLL_UP_L1:

        PUSH    AX

        CALL    L_SCROLL_UP_1

        POP     AX

        DEC     AL

        JNZ     LAVA_SCROLL_UP_L1

        JMP     VIDEO_RETURN

;-----

                                ;AH = 07h    VIDEO - SCROLL DOWN WINDOW

SCROOL_DOWN:                    ;AL = number of lines by which to scroll up (00h = clear entire window)

        CMP     word [CONSOL_FLAG],0    ;Send output to Propeller board (0), Video board (1), or LAVA Board (2)

        JZ      PROPELLER_SCROOL_DOWN

        CMP     word [CONSOL_FLAG],2    ;Send output to Propeller board (0), Video board (1), or LAVA Board (2)

        JZ      LAVA_SCROOL_DOWN

        JMP     VGA_SCROOL_DOWN

```

PROPELLER_SCROLL_DOWN:

```

    OR     CX,CX                ;Start 0,0?, (CH,CL = row,column start, DH,DL = row,column end)
    JZ     SCROLL_DOWN_0
    PUSH   AX
    PUSH   DX
    MOV    DX,CX                ;At least we will reposition cursor
    CALL   PROPELLER_SET_CURSOR ;Set Cursor at [DX]
    POP    DX
    POP    AX
    JMP    VIDEO_NOT_FINISHED ;Will ignore DX for now

```

SCROLL_DOWN_0:

```

    CMP    AL,0                ;AL has number of lines to scroll
    JNZ    SCROLL_DOWN_1
    MOV    AL,40                ;0 for current 40 line CRT

```

SCROLL_DOWN_1:

```

    PUSH   AX
    MOV    AH,ESC
    CALL   FAST_CONOUT
    MOV    AH,'M'
    CALL   FAST_CONOUT
    POP    AX
    DEC    AL
    JNZ    SCROLL_DOWN_1
    JMP    VIDEO_RETURN

```

LAVA_SCROLL_DOWN:

```

    OR     CX,CX                ;Start 0,0?, (CH,CL = row,column start, DH,DL = row,column end)
    JZ     SCROLL_DOWN_L0
    PUSH   AX
    PUSH   DX
    MOV    DX,CX                ;At least we will reposition cursor
    CALL   L_SET_CURSOR         ;Set Cursor at [DX]
    POP    DX
    POP    AX
    JMP    VIDEO_NOT_FINISHED   ;Will ignore DX for now

```

SCROLL_DOWN_L0:

```

    CMP    AL,0                 ;AL has number of lines to scroll
    JNZ    SCROLL_DOWN_L1
    CALL   L_CLEAR_SCREEN       ;AL=0 then clear whole screen
    JMP    VIDEO_RETURN

```

SCROLL_DOWN_L1:

```

    JMP    VIDEO_NOT_FINISHED   ;<<<< Will ignore DX for now

```

```

    PUSH   AX

```

```

;    CALL   L_SCROLL_DOWN       ;<<<<< Not Done Yet

```

```

    POP    AX

```

```

    DEC    AL

```

```

    JNZ    SCROLL_DOWN_L1

```

```

    JMP    VIDEO_RETURN

```



```
;-----
```

```
;AH = 08h VIDEO - READ CHARACTER AND ATTRIBUTE AT CURSOR POSITION
```

```
READ_CHAR_ATT: ;AL = character to display
```

```
CMP word [CONSOL_FLAG],0 ;Send output to Propeller board (0), Video board (1), or LAVA Board (2)
```

```
JZ PROPELLER_READ_CHAR_ATT
```

```
CMP word [CONSOL_FLAG],2
```

```
JZ LAVA_READ_CHAR_ATT
```

```
JMP VGA_READ_AC_CURRENT
```

```
PROPELLER_READ_CHAR_ATT: ;AL = character to display
```

```
OR BH,BH ;BH = page number (00h to number of pages - 1)
```

```
JNZ VIDEO_NOT_FINISHED ;BL = attribute (text mode) or color (graphics mode)
```

```
MOV AH,07 ;Return: AH = character's attribute (text mode only)
```

```
MOV AL,0 ; AL = character (Not implemented)
```

```
JMP VIDEO_RETURN
```

```
LAVA_READ_CHAR_ATT: ;AL = character to display
```

```
OR BH,BH ;BH = page number (00h to number of pages - 1)
```

```
JNZ VIDEO_NOT_FINISHED ;BL = attribute (text mode) or color (graphics mode)
```

```
MOV AH,07 ;Return: AH = character's attribute (text mode only)
```

```
MOV AL,0 ; AL = character (Not implemented)
```

```
JMP VIDEO_RETURN
```

```
;-----
```

```
;AH = 09h VIDEO - WRITE CHARACTER AND ATTRIBUTE AT CURSOR POSITION
```

```
WRITE_AT_CURSOR_ATT: ;AL = character to display
```

```
    CMP    word [CONSOL_FLAG],0 ;Send output to Propeller board (0), Video board (1), or LAVA Board (2)
```

```
    JZ     PROPELLER_WRITE_AT_CURSOR_ATT
```

```
    CMP    word [CONSOL_FLAG],2
```

```
    JZ     LAVA_WRITE_AT_CURSOR_ATT
```

```
    JMP    VGA_WRITE_AC_CURRENT ;Send to display on CGA/VGA Board
```

```
PROPELLER_WRITE_AT_CURSOR_ATT:
```

```
    OR     BH,BH ;BH = page number (00h to number of pages - 1)
```

```
    JNZ    VIDEO_NOT_FINISHED ;BL = attribute (text mode) or color (graphics mode)
```

```
    MOV    AH,AL ;CX = number of times to write character
```

```
AT_CURSOR1:
```

```
    CALL   FAST_CONOUT ;Fast direct output to Propeller board
```

```
    LOOP   AT_CURSOR1 ;Repeat CX times
```

```
    JMP    VIDEO_RETURN
```

```
LAVA_WRITE_AT_CURSOR_ATT:
```

```
    OR     BH,BH ;BH = page number (00h to number of pages - 1)
```

```
    JNZ    VIDEO_NOT_FINISHED ;BL = attribute (text mode) or color (graphics mode)
```

```
    MOV    AH,AL ;CX = number of times to write character
```

```
L_AT_CURSOR:
```

```
    CALL   L_TTY_OUT ;Fast direct output to LAVA board (Character in AH)
```

```
    LOOP   L_AT_CURSOR ;Repeat CX times
```

```
    JMP    VIDEO_RETURN
```

```

;-----
;AH = 0Ah VIDEO - WRITE CHARACTER ONLY AT CURSOR POSITION

WRITE_AT_CURSOR:
;AL = character to display

    CMP    word [CONSOL_FLAG],0      ;Send output to Propeller board (0), Video board (1), or LAVA Board (2)

    JZ     PROPELLER_WRITE_AT_CURSOR

    CMP    word [CONSOL_FLAG],2

    JZ     PROPELLER_WRITE_AT_CURSOR

    JMP    VGA_WRITE_C_CURRENT ;Send to display on CGA/VGA Board

PROPELLER_WRITE_AT_CURSOR:

    OR     BH,BH                    ;BH = page number (00h to number of pages - 1)

    JNZ    VIDEO_NOT_FINISHED ;BL = attribute (text mode) or color (graphics mode)

    MOV    AH,AL                    ;CX = number of times to write character

AT_CURSOR2:

    CALL   FAST_CONOUT              ;Fast direct output to Propeller board

    LOOP   AT_CURSOR2              ;Repeat CX times

    JMP    VIDEO_RETURN

LAVA_WRITE_AT_CURSOR:

    OR     BH,BH                    ;BH = page number (00h to number of pages - 1)

    JNZ    VIDEO_NOT_FINISHED ;BL = attribute (text mode) or color (graphics mode)

    MOV    AH,AL                    ;CX = number of times to write character

AT_CURSOR2L:

    CALL   L_TTY_OUT                ;Fast direct output to LAVA board (Character in AH)

    LOOP   AT_CURSOR2L              ;Repeat CX times

```

```

        JMP     VIDEO_RETURN

;-----

                                ;AH = 0EH      ***** Simple TTY Output mode *****
VIDEO_TTY:                                ;AL = character, BL Background (1= green), BH = page

        TEST    byte[KB_FLAG_1],HOLD_STATE ;Is Pause flag set

        JNZ     VIDEO_TTY

                                ;Send output to Propeller board (0), Video board (1), or LAVA Board (2)

        CMP     word [CONSOL_FLAG],0

        JZ      PROPELLER_VIDEO_TTY

        CMP     word [CONSOL_FLAG],2

        JZ      LAVA_VIDEO_TTY

        JMP     VGA_WRITE_TTY

PROPELLER_VIDEO_TTY:

        IN      AL,KEYSTAT          ;Default Propeller or SD SYSTEMS VIDIO BOARD PORT

        AND     AL,4H              ;Is board ready for character

        JZ      VIDEO_TTY

        POP     AX                  ;<-- Get character from AX on stack above

        OUT     KEYOUT,AL

        JMP     VIDEO_RETURN1      ;Note the normal extra AX was removed from stack above

LAVA_VIDEO_TTY:

        MOV     AH,AL

        CALL    L_TTY_OUT          ;Fast direct output to LAVA board (Character in AH)

```

```
JMP    VIDEO_RETURN
```

```
;-----
```

```
                ;AH = 0Fh    VIDEO - GET VIDEO PARAMATERS
```

```
GET_VIDEO_PARMS:    ;AH = Number of CRT Columns, AL = Display Mode
```

```
    CMP    word [CONSOL_FLAG],0    ;Send output to Propeller board (0) or Video board (1), or LAVA Board (2)
```

```
    JZ     PROPELLER_GET_VIDEO_STATE
```

```
    CMP    word [CONSOL_FLAG],2
```

```
    JZ     LAVA_GET_VIDEO_STATE
```

```
    JMP    VGA_GET_VIDEO_STATE
```

```
PROPELLER_GET_VIDEO_STATE:
```

```
    MOV    AX,5003H
```

```
    MOV    BX,0    ;BH = Current page
```

```
    JMP    VIDEO_RETURN
```

```
LAVA_GET_VIDEO_STATE:
```

```
    MOV    AX,6303H    ;99 Characters/line
```

```
    MOV    BX,0    ;BH = Current page
```

```
    JMP    VIDEO_RETURN
```

```
;-----
```

```
                ;AH = 12H    VIDEO - ALTERNATE FUNCTION SELECT (VGA, MCGA) - VIDEO ADDRESSING
```

```
VIDEO_VGA:        ;BL = 32, return AL = new state. 0 = enabled, 1 = disable
```

```
    CMP    BL,32H    ;Use for Cirrus VGA Board initialization (only)
```

```

JNZ    VIDEO_TBD

MOV     AL,12H           ;Return AL = 12h if function is supported

JMP     VIDEO_RETURN

```

```

;-----

```

```

;AH = 13H      VIDEO - WRITE CHARACTER AND ATTRIBUTE AT CURSOR POSITION

```

```

WRITE_STRING:           ;ES:BP = string pointer

```

```

MOV     ES,[ES_STORE]   ;Saved at the start

CMP     word [CONSOL_FLAG],0      ;Send output to Propeller board (0), Video board (1), or LAVA Board (2)

JZ      PROPELLER_WRITE_STRING    ;CX = string length

CMP     word [CONSOL_FLAG],2

JZ      LAVA_WRITE_STRING   ;CX = string length

JMP     VGA_WRITE_STRING    ;DX = cursor position

```

```

PROPELLER_WRITE_STRING:

```

```

OR      BH,BH           ;Can do only page 0

JNZ     VIDEO_NOT_FINISHED ;BH = page number (00h to number of pages - 1)

OR      CX,CX           ;For zero length string, just return (as IBM does)

JZ      PrW2

OR      AL,AL           ;AL = 0, do not move cursor, AL = 1, update cursor. AL=3 or 4 add attribute also

JZ      PrW1           ;Currently only AL=0 & 1 mode implemented

CALL    PROPELLER_SET_CURSOR      ;Set Cursor at [DX]

```

```

PrW1: MOV     AH,[ES:BP]      ;Send string to console

```

```

CALL    FAST_CONOUT

```

```

        INC     BP                ;No need to save BP

        LOOP   PrW1

PrW2:   JMP     VIDEO_RETURN

LAVA_WRITE_STRING:

        OR     BH,BH              ;Can do only page 0

        JNZ    VIDEO_NOT_FINISHED ;BH = page number (00h to number of pages - 1)

        OR     CX,CX              ;For zero length string, just return (as IBM does)

        JZ     LPrW2

        PUSH   AX                ;Save Cursor update info

        OR     AL,AL              ;AL = 0, do not move cursor, AL = 1, update cursor. AL=3 or 4 add attriute also

        JZ     LPrW1

        CALL   L_HIDE_CURSOR

LPrW1:  MOV     AH,[ES:BP]         ;Send string to console

        CALL   L_TTY_OUT_NO_UPDATE

        INC     BP                ;No need to save BP

        LOOP   LPrW1

        POP    AX

        OR     AL,AL              ;AL = 0, do not move cursor, AL = 1, update cursor. AL=3 or 4 add attriute also

        JZ     LPrW2

        CALL   L_NEXT_POSITION    ;Advance the cursor one position, next line if at EOL, Scroll up if at bottom of screen

        CALL   L_SHOW_CURSOR      ;Show new cursor position

LPrW2:  JMP     VIDEO_RETURN

```

```

;----- CGA/VGA Video board routines -----
;
;   Note all INT 10H generated Console outputs will come here if [CONSOLE_FLAG] = 1
;
VGA_SET_MODE:
    CALL    VGA_INIT          ;Want this callable because it is used at Initilization
    JMP     VIDEO_RETURN

                                ;Arrive here with ES: set to Video RAM area, DS:=0, AL=Mode
VGA_INIT:                    ;Initilize the S-100 Lomas CGA video board (or compatible board)
    mov     dx,c6845port+4      ;address of colour card
    mov     bl,0                ;mode set for colour
    cmp     di,30h
    jne     m8
    mov     al,7
    mov     dx,bw6845port+4     ;address of b/w card
    inc     bl                  ;mode set for bw card

m8:  mov     ah,al              ;save mode in ah
    mov     [CRT_MODE],al      ;store it
    mov     [ADDR_6845],dx     ;also chip ports

    PUSH    DS                 ;Stuff in capitals below are mods in the AT-BIOS ROM
    push    ax

```



```

push    dx                ;save 6845 base reg

add     dx,4              ;point to control reg = 3d8h
mov     al,bl             ;get mode
out     dx,al             ;setup chip for new mode

pop     dx                ;back to base 6845 reg
sub     ax,ax
mov     ds,ax             ;DS: to 0

LDS     bx,[VID_PARM_PTR] ;DS=0, BX=VID_PARM_PTR, will set DS=CS (here), BX to VID_PARM_TABLES
pop     ax                ;get back parameters

mov     cx,Index_Reg_Count ;length of row table
cmp     ah,2              ;Need to figure out which patamater table to use
jc      m9                ;mode 0 or 1 (40X25)
add     bx,cx              ;go to next row of int table, ie 16 bytes higher
cmp     ah,4
jc      m9                ;mode 2 or 3 (80X25)
add     bx,cx
cmp     ah,7
jc      m9                ;mode is 4,5 or 6 (graphics mode)
add     bx,cx              ;else BW Card paramaters

m9:     push    ax          ;BX points to correct row of init Table

```

```

PUSH    ES

XOR     AX,AX                ;Not clear why AT BIOS has this stuff

MOV     ES,AX                ;ES:-> 0

MOV     AX,[BX+10]           ;Uses DS:=0

XCHG    AH,AL

MOV     [ES:CURSOR_MODE],AX

POP     ES

        xor     ah,ah

m10:    mov     al,ah         ;Block output iniation parms to chip.

        out     dx,al

        inc     dx           ;next port

        inc     ah           ;next value

        mov     al,[bx]      ;Note this will use [DS:BX] from the above LDS

        out     dx,al

        inc     bx

        dec     dx           ;back to pointer reg

        loop    m10

        pop     ax

        POP     DS

```

```

xor    di,di                ;fill video with blanks
mov     [CRT_START],di      ;start address
mov     byte[ACTIVE_PAGE],0
mov     cx,8192              ;number of words in colour card
cmp     ah,4                ;Test for colour card
jc      m12
cmp     ah,7
je      m11
xor     ax,ax                ;fill for graphics mode
jmp     short m13
m11:    mov     CH,08H        ;buffer size for b/w card
m12:    mov     ax,' '+ (02H*256) ;<<<<NOTE IBM USES WHITE ON BLACK (07), we use 02,Green
m13:    rep     stosw         ;AX -> [ES:DI] (Normally B800:0 upwards)

mov     al,[CRT_MODE]        ;;;get the mode ->AL
xor     ah,ah                ;AH to 0
mov     si,ax
mov     dx,[ADDR_6845]
add     dx,4

mov     al,[CS:si+m7]        ;Make sure we use the CS: override (DS:=0)

out     dx,al
mov     [CRT_MODE_SET],al    ;save that value

```

```

mov     al,[CS:si+m6]
xor     ah,ah
mov     [CRT_COLS],ax

and     si,0eh                ;word offset into clear length table
mov     cx,[CS:si+m5]        ;Make sure we use teh CS: override (DS:=0)
mov     [CRT_LEN],cx         ;save length of crt
mov     cx,2                  ;clear all cursor positions (we have space for only 2)
mov     di,[CURSOR_POSN]
push    ds                    ;DS=0
pop     es                    ;ES=0
xor     ax,ax                 ;AX=0 -> [ES:DI]
rep     stosw                 ;Repeat 2 times

inc     dx                    ;Set overscan port to a default
mov     al,30h
cmp     byte[CRT_MODE],6
jnz     m14
mov     al,3fh                ;640 x 200 is special case
m14:    out     dx,al
mov     [CRT_PALETTE],al     ;store value
ret

```

```
;      SET CURRENT CURSOR VALUE

;      CX = CURSOR VALUE, CH (BITS 4-0) START LINE, CL (BITS (4-0) STOP LINE
```

```
VGA_SET_CURSOR_TYPE:                ;AH = 1

    mov     ah,10                    ;set cursor value
    mov     [CURSOR_MODE],cx
    call    m16
    JMP     VIDEO_RETURN

m16:  mov     dx,[ADDR_6845]          ;this routine outputs cx to 6845 reg in ah
    mov     al,ah
    out     dx,al
    inc     dx
    JMP     short+$+2                ;IO delay (AT-BIOS)
    mov     al,ch
    out     dx,al
    dec     dx
    JMP     short+$+2                ;IO delay (AT-BIOS)
    mov     al,ah
    inc     al                        ;point to other data reg
    out     dx,al
    inc     dx
    JMP     short+$+2                ;IO delay (AT-BIOS)
    mov     al,cl
    out     dx,al
```

```
ret
```

```
;      SET CURRENT CURSOR POSITION TO NEW X-Y VALUE
;
;      DX = ROW,COLUMN OF NEW CURSOR
;
;      BH = DISPLAY PAGE OF NEW CURSOR (MUST BE 0 FOR GRAPHICS MODE)
```

```
VGA_SET_CURSOR_POS:          ;AH = 2
    mov     cl,bh              ;get display page
    xor     ch,ch
    sal     cx,1
    mov     si,cx
    mov     [si+Cursors_POSN],dx ;DS=0
    cmp     [ACTIVE_PAGE],BH
    jnz     m17                ;if not current page abort
    mov     ax,dx
    call    m18
m17:  JMP     VIDEO_RETURN

m18:  call    POSITION          ;Set Cursor pos. AX has Row/Col info
    mov     cx,ax
    add     cx,[CRT_START]
    sar     cx,1
    mov     ah,14              ;reg no of cursor
    call    m16
```

```
ret
```

```
;      READ CURSOR POSITION
```

```
;      BH = PAGE OF CURSOR
```

```
;output:-
```

```
;      DX = ROW,COLUMN OF THE CURSOR POSITION
```

```
;      CX = CURRENT CURSOR MODE
```

```
VGA_GET_CURSOR_POS:          ;AH = 03h      VIDEO - GET CURSOR POSITION AND SIZE
```

```
mov    bl,bh
```

```
xor    bh,bh
```

```
sal    bx,1
```

```
mov    dx,[bx+CURSOR_POSN]
```

```
mov    cx,[CURSOR_MODE]
```

```
POP    AX                      ;Remove the "extra AX"
```

```
POP    BP                      ;New for AT-BIOS
```

```
POP    DI
```

```
POP    SI
```

```
POP    BX
```

```
POP    AX                      ;Discard old CX & DX
```

```
POp    AX
```

```
POP    DS
```

```
POP    ES
```

```
IRET
```

```
;      SET THE ACTIVE DISPLAY PAGE
```

```
;      AL = NEW DISPLAY PAGE
```

```
VGA_SET_PAGE:                ;AH = 5
```

```
    mov     [ACTIVE_PAGE],al
```

```
    mov     cx,[CRT_LEN]
```

```
    cbw
```

```
    push    ax
```

```
    mul     cx
```

```
    mov     [CRT_START],ax
```

```
    mov     cx,ax
```

```
    sar     cx,1
```

```
    mov     ah,12
```

```
    call    m16
```

```
    pop     bx
```

```
    sal     bx,1
```

```
    mov     ax,[bx+CURSOR_POSN]
```

```
    call    m18
```

```
    JMP     VIDEO_RETURN
```



```
;      SET BACKGROUND COLOUR,OVERSCAN COLOUR, AND FORGROUND COLOUR FOR MEDIUM RESOLUTION GRAPHICS
;      BH = 0 THEN BACKGROUND SET FROM LOW BITS OF BL (0-31)
;      BH = 1 THE PALLET SELECTION IS MADE BASED ON LOW BITS OF BL:-
;
;          0 = GREEN, RED,YELLOW FOR COLOURS 1,2,3
;          1 = BLUE,CYAN,MAGENTA FOR COLOURS 1,2,3
;
;      BL = COLOUR VALUE TO BE USED
```

```
VGA_SET_COLOR:                                ;AH = 0BH
    mov     dx,[ADDR_6845]
    add     dx,5
    mov     al,[CRT_PALETTE]
    or      bh,bh
    jnz     m20

    and     al,0e0h                            ;handle for colour 0
    and     bl,1fh
    or      al,bl

m19:  out     dx,al
    mov     [CRT_PALETTE],al
    JMP     VIDEO_RETURN

m20:  and     al,0dfh                            ;handel colour 1
```

```

shr    bl,1
jnc    m19
or     al,20h
jmp    m19

```

```

;      GET CURRENT VIDIO STATE
;      AL = CURRENT VIDIO MODE
;      BH = CURRENT ACTIVE PAGE

```

```

VGA_GET_VIDEO_STATE:                ;AH = 0FH

mov    ah,[CRT_COLS]
mov    al,[CRT_MODE]
mov    bh,[ACTIVE_PAGE]

POP    DI                          ;Remove the "extra AX"
POP    BP                          ;New for AT-BIOS
POP    DI
POP    SI
POP    CX                          ;Discard saved BX
JMP    M15                         ;Jump to remainder on VIDEO_RETURN

```

```

;      Scroll up text on screen

```

```

;      AL      =      NUMBER OF ROOLS TO SCROOL
;
;      CX      =      ROW/COL UPPER LEFT
;
;      DX      =      ROW/COL LOWER RIGHT
;
;      BH      =      ATTRIBUTE
;
;      DS      =      DATA SEGMENT
;
;      ES      =      VIDEO BUFFER

```

```

VGA_SCROOL_UP:                                ;AH = 6
;
;      CALL    TEST_LINE_COUNT                ;New in AT-BIOS
;
;      cmp     ah,4
;
;      jc      n1
;
;      cmp     ah,7
;
;      je      n1
;
;      jmp     graphics_up

n1:      push   bx
;
;      mov     ax,cx
;
;      call    scrool_position
;
;      jz      n7
;
;      add     si,ax
;
;      mov     ah,dh
;
;      sub     ah,b1

n2:      call   n10
;
;      add     si,bp

```

```

        add    di, bp
        dec    ah
        jnz    n2

n3:     pop     ax
        mov     al, ' '           ;fill with blanks

n4:     call    n11
        add     di, bp
        dec     bl
        jnz     n4

                               ; (DDS) the lomas board does not need video on here

n5:     xor     ax, ax           ;get location 0
        mov     ds, ax
        cmp     byte [CRT_MODE], 7
        je     n6
        mov     al, [CRT_MODE_SET]
        mov     dx, 3d8h         ;set colour port
        out     dx, al

n6:     JMP     VIDEO_RETURN

n7:     mov     bl, dh
        jmp     n3

```

;the lomas board does not need video off here

scrool_position: ;also no need to wait for retrace

cmp byte [CRT_MODE],2

jnb n9

cmp byte [CRT_MODE],3

ja n9

push dx ;must be 80x25 colour scroll

mov dx,3dah

push ax

n8: in al,dx

test al,8 ;wait for vertical retrace

jz n8

mov al,25h

mov dx,3d8h

out dx,al ;turn off vidio

pop ax ;during retrace

pop dx

n9: call POSITION

add ax,[CRT_START]

mov di,ax

mov si,ax

sub dx,cx

```

inc    dh
inc    dl
xor     ch,ch
mov     bp,[CRT_COLS]
add     bp,bp
mov     al,bl
mul     byte [CRT_COLS]
add     ax,ax
push    es
pop     ds
cmp     bl,0
ret

```

```

n10:   mov     cl,dl                ;no of columns to move
       push    si
       push    di
       rep movsw
       pop     di
       pop     si
       ret

```

```

n11:   mov     cl,dl                ;no of columns to clear
       push    di
       rep     stosw
       pop     di

```

```
ret
```

```
;      Scroll down text on screen
;      Ah      =      CURRENT CRT MODE
;      AL      =      NUMBER OF ROOLS TO SCROOL
;      CX      =      ROW/COL UPPER LEFT
;      DX      =      ROW/COL LOWER RIGHT
;      BH      =      ATTRIBUTE
;      DS      =      DATA SEGMENT
;      ES      =      VIDEO BUFFER
;
```

```
VGA_SCROOL_DOWN:          ;AH = 7
    std
    CALL  TEST_LINE_COUNT      ;New in AT-BIOS
    cmp   ah,4
    jc    n12
    cmp   ah,7
    je    n12
    jmp   graphics_down
```

```
n12:  push  bx
```

```
    mov     ax,dx
    call    scrool_position
    jz      n16
    sub     si,ax
    mov     ah,dh
    sub     ah,b1

n13:  call    n10
    sub     si,bp
    sub     di,bp
    dec     ah
    jnz     n13

n14:  pop     ax
    mov     al,' '

n15:  call    n11
    sub     di,bp
    dec     bl
    jnz     n15
    jmp     n5

n16:  mov     bl,dh
    jmp     n14
```



```

TEST_LINE_COUNT:                ;New in AT-BIOS

    MOV     BL,AL                ;If lines to be scrolled = lines in window, adjust AL else return
    OR      AL,AL
    JZ      BL_SET
    PUSH    AX
    MOV     AL,DH
    SUB     AL,CH
    INC     AL
    CMP     AL,BL
    POP     AX
    JNE     BL_SET
    SUB     BL,BL

BL_SET RET

```

```

;      READ CURRENT CHARACTER AND ATTRIBUTE
;
;      AH = CURRENT CRT MODE
;
;      BH = DISPLAY PAGE
;
;      DS = DATA SEGMENT
;
;      ES = REGEN SEGMENT
;
;output:-
;
;      AH = ATTRIBUTE READ
;
;      AL = CHAR READ
;
;
VGA_READ_AC_CURRENT:            ;AH = 08H

```

```

    cmp     ah,4
    jc      p1
    cmp     ah,7
    je      p1
    jmp     graphics_read

p1:    call  find_position
    mov     si,bx

    mov     dx,[ADDR_6845]          ;wait for retreace
    add     dx,6
    push    es
    pop     ds

p2:    in     al,dx
    test    al,1
    jnz     p2
    cli                      ;no more ints

p3:    in     al,dx
    test    al,1
    jz      p3
    lodsw

    JMP     VIDEO_RETURN

```

```
find_position:
```

```
    mov     cl,bh
    xor     ch,ch
    mov     si,cx
    sal     si,1
    mov     ax,[si+Cursors_Posn]
    xor     bx,bx
    jcxz    p5
```

```
p4:    add     bx,[CRT_LEN]
        loop   p4
```

```
p5:    call    POSITION
        add     bx,ax
        ret
```

```
;      WRITE  CHAR AND ATTRIBUTE AT CURRENT CURSOR POSITION
;
;      AH = CURRENT CRT MODE
;
;      BH = DISPLAY PAGE
;
;      CX = COUNT OF CHARACTERS TO WRITE
;
;      AL = CHAR TO WRITE
;
;      BL = ATTRIBUTE OF CHAR TO WRITE
;
;      DS = DATA SEGMENT
```

```
;      ES = REGEN SEGMENT
```

```
VGA_WRITE_AC_CURRENT:      ;AH = 9
```

```
    cmp     ah,4
    jc      p6
    cmp     ah,7
    je      p6
    jmp     graphics_write
```

```
p6:  mov     ah,b1
      push   ax
      push   cx
      call   find_position
      mov    di,bx
      pop    cx
      pop    bx
```

```
p7:  mov     dx,[ADDR_6845]
      add    dx,6
```

```
p8:  in      al,dx
      test   al,1
      jnz    p8
      cli
```

```

p9:   in     al,dx

      test  al,1

      jz    p9

      mov   ax,bx

      stosw

      sti

      loop  p7

      JMP   VIDEO_RETURN

```

```

;      WRITE CHAR AT CURSOR POSITION DO NOT CHANGE ATTRIBUATE

;      BH = DISPLAY PAGE

;      CX = COUNT OF CHARACTERS TO WRITE

;      AL = CHAR TO WRITE

;      DS = DATA SEGMENT

;      ES = REGEN SEGMENT

```

```

VGA_WRITE_C_CURRENT:                ;AH = 0AH

      cmp   ah,4

      jc    p10

      cmp   ah,7

      je    p10

      jmp   graphics_write

```

```
p10:  push  ax
      push  cx
      call  find_position
      mov   di,bx
      pop   cx
      pop   bx

p11:   mov   dx,[ADDR_6845]
      add   dx,6

p12:   in     al,dx
      test  al,1
      jnz   p12
      cli

p13:   in     al,dx
      test  al,1
      jz    p13
      mov   al,b1
      stosb
      sti
      inc   di                ;go past attribuate
      loop  p11
      JMP   VIDEO_RETURN
```

```

;AH = 13H      VIDEO - WRITE CHARACTER AND ATTRIBUTE AT CURSOR POSITION

```

```

VGA_WRITE_STRING:    ;ES:BP = string pointer (Note: New in AT-BIOS)

```

```

    CMP    AL,04      ;Test for invalid request

```

```

    JB     W0

```

```

    JMP    DONE

```

```

W0:   OR     CX,CX      ;Return if zero length

```

```

    JNZ    W1

```

```

    JMP    DONE

```

```

W1:   PUSH   BX

```

```

    MOV    BL,BH        ;get current cursor position for that page

```

```

    XOR    BH,BH

```

```

    SAL    BX,1          ;X2

```

```

    MOV    SI,[BX+Cursors_Posn] ;0,1,2,..UP TO 8 PAGES

```

```

    POP    BX

```

```

    PUSH   SI            ;save current cursor position

```

```

    PUSH   AX            ;save write string option

```

```

    MOV    AX,0200H

```

```

    INT    10H          ;set new cursor position

```

```

    POP    AX

```

WRITE_CHAR:

PUSH CX

PUSH BX

PUSH AX

PUSH ES

XCHG AH,AL

MOV AL,[ES:BP]

INC BP

CMP AL,08H ;special cases, BS

JE DD_TTY

CMP AL,0DH

JE DD_TTY

CMP AL,0AH

JE DD_TTY

CMP AL,07H

JNE GET_ATTRIBUTE

DD_TTY: MOV AH,0EH ;write to tty

INT 10H

MOV BL,BH

SAL BH,1 ;X2

MOV DX,[BX+CURLSOR_POSN]

POP ES


```

POP     AX
POP     BX
POP     CX
JMP     ROWS_SET

```

GET_ATTRIBUTE:

```

MOV     CX,1

CMP     AH,2           ;If AL is 1 or 2 then AL has ASCII character, BL has the Attribute
JB      GOT_IT         ;If 3 or 4 then attrib, char, attrtrib, char.....

MOV     BL,[ES:BP]

INC     BP

```

```

GOT_IT:  MOV     BH,0

MOV     CX,1

MOV     AH,09H         ;write char & attribute on crt

INT     10H

POP     ES

POP     AX

POP     BX

POP     CX

INC     DL             ;inc column count

CMP     DL,[CRT_COLS]

JB      COLUMNS_SET

```

```
INC    DH
SUB    DL,DL
CMP    DH,25          ;bottom of page
JB     ROWS_SET

PUSH   ES
PUSH   AX
MOV    AX,0E0AH       ;scroll down one line
INT    10H
DEC    DH
POP    AX
POP    ES
```

ROWS_SET:

COLUMNS_SET:

```
PUSH   AX
MOV    AX,0200H       ;set cursor position
INT    10H
POP    AX
LOOP   WRITE_CHAR

POP    DX
CMP    AL,1
JE     DONE
CMP    AL,3
```

```

        JE      DONE
        MOV     AX,0200H          ;set cursor position
DONE:   JMP     VIDEO_RETURN

```

```

;      READ OR WRITE A DOT AT INDICATED POSITION
;
;      DX = ROW (0-199)
;
;      CX = COLUMN (0-639)
;
;      AL = DOT VALUE (see text)
;
;      DS = DATA SEGMENT
;
;      ES = REGEN SEGMENT
;output:-
;
;      AL = DOT VALUE READ, RIGHT JUSTIFIED

```

```

VGA_READ_DOT:                ;AH = 0DH
        call    RX3
        mov     al,[es:si]
        and     al,ah
        shl     al,cl
        mov     cl,dh
        rol     al,cl
        JMP     VIDEO_RETURN

```

```

VGA_WRITE_DOT:                                ;AH = 0CH

    push    ax
    push    ax
    call    RX3
    shr     al,cl
    and     al,ah
    mov     cl,[es:si]
    pop     bx
    test    bl,80h
    jnz     rx2
    not     ah
    and     cl,ah
    or      al,cl

rx1:  mov     [es:si],al
    pop     ax
    JMP     VIDEO_RETURN

rx2:  xor     al,cl
    jmp     rx1

;
;      THIS ROUTINE DETERMINES THE RAM LOCATION OF COL/ROW
;input:-

```

```

;      DX = ROW (0-199)

;      CX = COLUMN (0-639)

;output:-

;      SI = OFFSET INTO RAM

;      AH = MASK TO STRIP OF BITS OF INTREST

;      CL = BITS TO SHIFT TO RIGHT JUSTIFY MASK IN AH

;      DH = NO OF BITS IN RESULT

;

RX3:  push  bx

      push  ax

      mov   al,40h

      push  dx

      and   dl,0FEH

      mul   dl

      pop   dx

      test  dl,1

      jz    rx4

      add   ax,2000H

rx4:  mov    si,ax

      pop   ax

      mov   dx,cx

```

```

    mov     bx,2C0H                ;determine graphics mode currently in effect
    mov     cx,302h
    cmp     byte [CRT_MODE],6
    jc      rx5
    mov     bx,180h
    mov     cx,703h

rx5:   and     ch,dl

        shr     dx,cl
        add     si,dx
        mov     dh,bh

        sub     cl,cl
rx6:   ror     al,1

        add     cl,ch
        dec     bh
        jnz     rx6

    mov     ah,bl
    shr     ah,cl
    pop     bx
    ret

```

```

;      GRAPHICS SCROLL UP
;      CH,CL = UPPER LEFT HAND CORNER OF SCREEN
;      DH,DL = LOWER RIGHT HAND CORNER OF SCREEN
;      BH = FILL CHAR FOR BLANK LINES
;      DS = DATA SEGMENT
;      ES = REGEX SEGMENT

graphics_up:
    mov     bl,al           ;save line count
    mov     ax,cx

    call    graph_posn
    mov     di,ax

    sub     dx,cx
    add     dx,101h
    sal     dh,1

    sal     dh,1

    cmp     byte [CRT_MODE],6
    jnc     rx7

```

```

        sal    dl,1
        sal    dl,1

rx7:    push    es
        pop     ds
        sub     ch,ch
        sal     bl,1
        sal     bl,1
        jz      rx11
        mov     al,bl
        mov     ah,80
        mul     ah
        mov     si,di
        add     si,ax
        mov     ah,dh
        sub     ah,bl

rx8:    call    rx17
        sub     si,2000h-80      ;next row
        sub     di,2000h-80
        dec     ah
        jnz     rx8

rx9:    mov     al,bh            ;fill in vacent lines
rx10:   call    RX18

```



```

sub    di,2000h-80
dec    bl
jnz    rx10
JMP    VIDEO_RETURN

```

```

rx11:  mov    bl,dh
      jmp    rx9

```

```

;      GRAPHICS SCROOL DOWN
;
;      CH,CL = UPPER LEFT HAND CORNER OF SCREEN
;
;      DH,DL = LOWER RIGHT HAND CORNER OF SCREEN
;
;      BH = FILL CHAR FOR BLANK LINES
;
;      DS = DATA SEGMENT
;
;      ES = REGEN SEGMENT

```

```

graphics_down:

```

```

      std
      mov    bl,al
      mov    ax,dx

      call   graph_posn
      mov    di,ax

```

```
sub    dx,cx
add    dx,101h
sal    dh,1
sal    dh,1

cmp    byte [CRT_MODE],6
jnc    rx12

sal    dl,1
sal    di,1
inc    di

rx12:  push  es
pop    ds
sub    ch,ch
add    di,240
sal    bl,1
sal    bl,1
jz     rx16
mov    al,bl
mov    ah,80
mul    ah
mov    si,di
sub    si,ax
mov    ah,dh
```

```
    sub    ah,b1

rx13: call  rx17
    sub    si,2000h+80
    sub    di,2000h+80
    dec    ah
    jnz    rx13

rx14: mov    al,bh

rx15: call  RX18
    sub    di,2000h+80
    dec    bl
    jnz    rx15
    cld
    JMP     VIDEO_RETURN

rx16: mov    bl,dh
    jmp     rx14

rx17: mov    cl,dl
    push    si
    push    di
    rep     movsb
```

```
pop    di
pop    si
add    si,2000h
add    di,2000h
push   si
push   di
mov    cl,dl
rep    movsb
pop    di
pop    si
ret
```

RX18: mov cl,dl

```
push   di
rep    stosb
```

```
pop    di
add    di,2000h
push   di
mov    cl,dl
rep    stosb
```

```
pop    di
ret
```

```
;    GRAPHICS WRITE
;    AL = CHAR TO WRITE
;    BL = COLOUR
;    CX = NO OF CHARACTERS
;    DS = DATA SEGMENT
;    ES = REGEN SEGMENT
```

```
graphics_write:
```

```
    mov    ah,0
```

```
    push   ax
```

```
    call   s26
```

```
    mov    di,ax
```

```
    pop     ax
```

```
    cmp    al,80h           ;is image in "rom" ie second half
```

```
    jae    s1
```

```
    mov    si,CRT_CHAR_GEN      ;Location of Image of characters in IBM Rom
```

```
    push   cs                  ;<<<< NOTE FOR 27128K EPROM or 8088 Board I HAVE REMOVED THIS CHARACTER TABLE
```

```
    jmp     short s2           ;THERE IS NOT ENOUGH ROOM IN ROM >>>>
```

```
s1:    sub    al,80h           ;image in second half
```


stosb

lodsb

s5: mov [es:di+1ffffh],al

add di,79

dec dh

jnz s4

pop si

pop di

inc di

loop s3

JMP VIDEO_RETURN

s6: xor al,[es:di]

stosb

lodsb

xor al,[es:di+1ffffh]

jmp s5

s7: mov dl,bl ;high res write

sal di,1

call s19

s8: push di

```
    push    si
    mov     dh,4

s9:   lodsb

    call    s21
    and     ax,bx

    test    dl,80h
    jz      s10
    xor     ah,[es:di]
    xor     al,[es:di+1]

s10:  mov     [es:di],ah
    mov     [es:di+1],al
    lodsb
    call    s21
    and     ax,bx
    test    dl,80h
    jz      s11
    xor     ah,[es:di+2000h]
    xor     al,[es:di+2001h]

s11:  mov     [es:di+2000h],ah
    mov     [es:di+2001h],al
    add     di,80
```



```

dec    dh
jnz    s9
pop    si
pop    di
inc    di
inc    di
loop   s8
JMP    VIDEO_RETURN

```

```

;      GRAPHICS READ
;      NONE (0 IS ASSUMED FOR BACKGROUND COLOUR)
;output:-
;      AL = CHAR  (0 IF NONE THERE)

```

graphics_read:

```

call   s26
mov    si,ax
sub    sp,8
mov    bp,sp

cmp    byte [CRT_MODE],6
push   es
pop    ds

```

jc s13

mov dh,4

```
s12:  mov    al,[si]
      mov    [bp],al
      inc    bp
      mov    al,[si+2000h]
      mov    [bp],al
      inc    bp
      add    si,80
      dec    dh
      jnz    s12
      jmp    s15
```

```
s13:  sal    si,1
      mov    dh,4
```

```
s14:  call   s23
      add    si,2000h
      call   s23
      sub    si,2000h-80
      dec    dh
      jnz    s14
```

```
s15:  mov     di,CRT_CHAR_GEN           ;char gen in IBM Rom
      push   cs                       ;<<<< NOTE FOR 27128K EPROM OR 8088 BOARD I HAVE REMOVED THIS CHARACTER TABLE
      pop    es                       ;THERE IS NOT ENOUGH ROOM IN ROM. SELDOM USED ANYWAY>>>>
      sub    bp,8
      mov    si,bp
      cld
      mov    al,0

s16:  push   ss
      pop    ds
      mov    dx,128

s17:  push   si
      push   di
      mov    cx,8
      repe   cmpsb

      pop    di
      pop    si
      jz     s18
      inc    al
      add    di,8
      dec    dx
      jnz    s17
```

```

    cmp     al,0
    je      s18
    sub     ax,ax
    mov     ds,ax           ;set ds to 0

    les     di,[EXT_CHAR_PTR]
    mov     ax,es
    or      ax,di
    jz      s18
    mov     al,128
    jmp     s16

s18:  add     sp,8           ;<<<<<<<< Check!
      JMP     VIDEO_RETURN

s19:  and     bl,3
      mov     al,bl
      push    cx
      mov     cx,3

s20:  sal     al,1
      sal     al,1
      or      bl,al
      loop    s20

```

```
    mov     bh,b1
    pop     cx
    ret
```

```
s21:  push   dx
       push   cx
       push   bx
       sub    DX,DX
       mov    cx,1
```

```
s22:  mov     bx,ax
       and     bx,cx
       or      dx,bx
       shl     ax,1
       shl     cx,1
       mov     bx,ax
       and     bx,cx
       or      dx,bx
       shl     cx,1
```

```
    jnc     s22
    mov     ax,dx
    pop     bx
    pop     cx
    pop     dx
```

ret

s23: mov ah,[si]
 mov al,[si+1]
 mov cx,0c000h
 mov dl,0

s24: test ax,cx
 clc
 jz s25
 stc

s25: rcl dl,1
 shr cx,1
 shr cx,1
 jnc s24
 mov [bp],dl
 inc bp
 ret

s26: mov ax,[CURSOR_POSN]

graph_posn:

push bx
 mov bx,ax
 mov al,ah

```

mul    byte [CRT_COLS]
shl    ax,1
shl    ax,1
sub    bh,bh
add    ax,bx
pop    bx
ret

```

```

;      WRITE_TTY          <<<< MAIN VIDEO BOARD CHARACTER OUTPUT ROUTINE >>>>>>
;      AL = CHARACTER
;      BL = BACKGROUND CHAR IF IN GRAPHICS MODE

```

```

VGA_WRITE_TTY:                ;AH = 0EH
    push    ax
    push    ax
    mov     ah,3
    MOV     BH,[ACTIVE_PAGE]
    int     10h                ;DX now has current Cursor position
    pop     ax                ;Get character

    cmp     al,8                ;is it BS
    je      u8

```

```

    cmp     al,0dh                ;Is it CR
    je      u9
    cmp     al,0ah                ;Is it LF
    je      u10
    cmp     al,07                 ;Is it BELL
    je      u11

    mov     ah,10                 ;Write char on screen
    mov     cx,1                 ;1X
    int     10h

    inc     dl
    cmp     dl,byte [CRT_COLS]
    jnz     u7
    mov     dl,0
    cmp     dh,24
    jnz     u6

u1:  mov     ah,2                 ;Set Cursor
    int     10h                 ;;Difference on AT-BIOS (PC has BH=0)

    mov     al,[CRT_MODE]
    cmp     al,4
    jc      u2
    cmp     al,7

```



```
        mov     bh,0
        jne     u3

u2:     mov     ah,8                ;Read cursor
        int     10h
        mov     bh,ah

u3:     mov     ax,601h            ;Scroll up one line
        sub     cx,cx
        mov     dh,24
        mov     dl,byte [CRT_COLS]
        dec     dl

u4:     int     10h

u5:     pop     ax
        JMP     VIDEO_RETURN

u6:     inc     dh

u7:     mov     ah,2
        jmp     u4

u8:     cmp     dl,0
        je      u7
        dec     dl
```

```

        jmp     u7

u9:     mov     dl,0
        jmp     u7

u10:    cmp     dh,24
        jne     u6
        jmp     u1

u11:    mov     bl,2
        call    BELL1          ;send hardware bell
        jmp     u5

;----- VIDEO SUPPORT ROUTINES -----

VIDEO_RETURN:          ;Most (but not all) routines finish up here
        POP     DI          ;Remove the "extra AX on stack"

VIDEO_RETURN1:
        POP     BP
        POP     DI
        POP     SI
        POP     BX
M15:    POP     CX
        POP     DX

```

```

POP     DS
POP     ES
IRET                                ;Note IRET

```

```

FAST_CONOUT:                        ;Fast send Character (in AH) to Propeller board
    IN     AL,KEYSTAT               ;Propeller or SD SYSTEMS VIDIO BOARD PORT
    AND    AL,4H                    ;Is board ready for character
    JZ     FAST_CONOUT
    MOV    AL,AH
    OUT    KEYOUT,AL
    RET

```

```

PROPELLER_SET_CURSOR:              ;Set cursor location to DH & DL
    MOV    AH,ESC
    CALL   FAST_CONOUT              ;Send is VT100 Format "ESC [ row ; column H"
    MOV    AH,'['
    CALL   FAST_CONOUT

    MOV    AL,DH                    ;DH = row (00h is top)
    CALL   HEX_TO_BCD               ;(DX unaltered for below)
    PUSH   AX
    MOV    AH,AL
    ROR    AH,1
    ROR    AH,1

```

```

ROR    AH,1
ROR    AH,1
AND    AH,0FH
ADD    AH,30H           ;Convert to ASCII
CALL   FAST_CONOUT     ;Send ROW 10's digit
POP    AX
MOV    AH,AL
AND    AH,0FH           ;Low nibble
ADD    AH,30H           ;Convert to ASCII
CALL   FAST_CONOUT     ;Send ROW 1's digit

MOV    AH,',';'
CALL   FAST_CONOUT

MOV    AL,DL            ;DL = Column (00h is left)
CALL   HEX_TO_BCD

PUSH   AX
MOV    AH,AL
ROR    AH,1
ROR    AH,1
ROR    AH,1
ROR    AH,1
AND    AH,0FH
ADD    AH,30H           ;Convert to ASCII

```

```

CALL    FAST_CONOUT        ;Send ROW 10's digit
POP     AX
MOV     AH,AL
AND     AH,0FH              ;Low nibble
ADD     AH,30H              ;Convert to ASCII
CALL    FAST_CONOUT        ;Send ROW 1's digit
MOV     AL,AH

MOV     AH,'H'
CALL    FAST_CONOUT
RET

```

```

;Calculate th Video RAM address from row/column

```

```

POSITION:

```

```

;ax = row,column

```

```

push    bx                  ;calculates ram address
mov     bx,ax
mov     al,ah
mul     byte [CRT_COLS]
xor     bh,bh
add     ax,bx
sal     ax,1
pop     bx
ret

```

VIDEO_NOT_FINISHED:

```

    PUSH    BX

    MOV     BX,VID_PARM_TBD1_MSG      ;"Int 10H Video paramater routine not fully implemented"

    JMP     VIDEO_TBD1

```

VIDEO_TBD:

```

    PUSH    BX

    MOV     BX,VID_PARM_TBD_MSG ;"Int 10H Video paramater not yet implemented"

```

VIDEO_TBD1:

```

    CALL    SERIAL_PRINT_STRING

    POP     BX

    POP     AX                      ;Recover that extra AX on stack

    CALL    SERIAL_DISPLAY_REGISTERS ;Display Registers on serial port display (All registers retained)

    JMP     VIDEO_RETURN1          ;Remember we have removed that one extra AX on stack

```

; Input: AL = input number Output: AL = BCD

HEX_TO_BCD:

```

    pushf                      ; Save flags register

    push cx                    ; Save general-purpose regs

    push dx

    push ax

    sub ah, ah                 ; We don't want a high-order byte so we don't have a divide overflow

```

```

mov dl, 0Ah          ; Divide by 10
div dl               ; Unsigned divide. Quotient in al,remainder in ah.
mov dl, ah           ; Save remainder
mov ah, al           ; Move quotient (multiple of 10)
mov cl, 4            ; and shift into high nibble of al
shr ax, cl           ; (8086 imposes stupid restrictions on shr operands)
or al, dl            ; Set low nibble of al to remainder
pop dx               ; Recover ah (pulling its value into dx first)
mov ah,dh            ; restore cx, dx and flags
pop dx
pop cx
popf
ret                  ; All done.

```

```

;----- BASIC LAVA-10 CRT TREMINAL COMMANDS -----

```

```

;           Only AL register changed unless stated otherwise

```

```

INITILIZE_LAVA:          ;Clear Screen, Green/Black Cursor 0,0

MOV    AL,00000000B      ;Set to WRITE mode, no strobes etc
OUT    LavaStatus,AL     ;Send to lava status port (91H)
CALL   L_CLEAR_BUFFER    ;Clear a buffer LAVA RAM area for Clear line etc
CALL   L_CLEAR_SCREEN
CALL   L_HOME             ;Set cursor X,Y to 0,0
MOV    BX,L_GREEN_COLOR

```

```

CALL    L_SET_COLOR
CALL    L_SHOW_CURSOR
RET

```

```

L_TTY_OUT:                ;<< CORE FUNCTION >>Write 1 character (in [AH]) to current cursor X,Y position. Update cursor
                           ;Special treatment for control characters.
    CMP    AH,SPACE
    JGE    L_NOT_SEPCIAL

```

```

    CMP    AH,CR           ;First treat the special case sitations
    JNZ    L_NOT_CR
    CALL    L_DO_CR
    RET

```

```

L_NOT_CR:
    CMP    AH,LF
    JNZ    L_NOT_LF
    CALL    L_DO_LF
    RET

```

```

L_NOT_LF:
    CMP    AH,BS
    JNZ    L_NOT_BS
    CALL    L_BACK_SPACE
    RET

```

```

L_NOT_BS:
    CMP    AH,SCROLL       ;Scroll up one line with 01H
    JNZ    L_NOT_SCROLL

```



```

        CALL    L_SCROLL_UP_1
        RET

L_NOT_SCROLL:
        CMP     AH,CLEAR
        JNZ     L_NOT_CLEAR
        CALL    L_CLEAR_SCREEN
        RET

L_NOT_CLEAR:
        CMP     AH,BELL
        JNZ     L_NOT_BELL
        CALL    BELL1           ;Send Bell to Propeller Board
        RET

L_NOT_BELL:

L_NOT_SEPCIAL:           ;We need to also take care of DEL (7FH)
        CMP     AH,DELETE
        JNZ     L_NOT_DEL
        CALL    L_DEL_SPACE
        RET

L_NOT_DEL:           ;Else do "regular" ASCII characters
        CALL    L_HIDE_CURSOR

        MOV     AL,DRAW$TEXT      ;Send Draw Text Command
        CALL    L_PULSE$WR

```

```

MOV     AL,1                ;Send 1 character only
CALL    L_PULSE$WR
MOV     AL,AH
CALL    L_PULSE$WR          ;Send Ascii
MOV     AL,0
CALL    L_PULSE$WR          ;send Ascii X2 (So we have an even number of bytes sent)
L_TTY_DONE:
CALL    L_NEXT_POSITION     ;Advance the cursor one position, next line if at EOL, Scroll up if at bottom of screen
CALL    L_SHOW_CURSOR       ;Show new cursor position
RET

L_TTY_OUT_NO_UPDATE:       ;<< CORE FUNCTION >>Write 1 character (in [AH]) to current cursor X,Y position. NO Update
cursor
MOV     AL,DRAW$TEXT        ;Send Draw Text Command
CALL    L_PULSE$WR
MOV     AL,1                ;Send 1 character only
CALL    L_PULSE$WR
MOV     AL,AH
CALL    L_PULSE$WR          ;Send Ascii
MOV     AL,0
CALL    L_PULSE$WR          ;send Ascii X2 (So we have an even number of bytes sent)
RET

L_HOME:                    ;Cursor to Top left of screen

```

```

PUSH    BX
CALL    L_HIDE_CURSOR
XOR     BX,BX
CALL    L_SET_X
CALL    L_SET_Y
CALL    L_SHOW_CURSOR
POP     BX
RET

```

```

L_SET_CURSOR:                ;AH = 02h      VIDEO - SET CURSOR POSITION
                                ;DH = row (00h is top), DL = column (00h is left)
    PUSH    BX
    XOR     BX,BX
    MOV     BL,DL
    SHL     BX,1                ;X2
    SHL     BX,1                ;X4
    SHL     BX,1                ;X8      (8 Pixels/character)
    CALL    L_SET_X
    XOR     BX,BX
    MOV     BL,DH
    SHL     BX,1                ;X2
    SHL     BX,1                ;X4
    SHL     BX,1                ;X8
    SHL     BX,1                ;X16     (16 Pixels/character)

```

```

CALL    L_SET_Y
POP     BX
RET

```

```

;AH = 03h      VIDEO - GET CURSOR POSITION AND SIZE
;DH = row (00h is top), DL = column (00h is left)

```

```
L_GET_CURSOR:
```

```

    PUSH    BX

    CALL    L_GET_X

    SHR     BX,1           ;/2
    SHR     BX,1           ;/4
    SHR     BX,1           ;/8    (8 Pixels/character)

    MOV     DL,BL

    CALL    L_GET_Y

    SHR     BX,1           ;/2
    SHR     BX,1           ;/4
    SHR     BX,1           ;/8

    SHR     BX,1           ;/16   (16 Pixels/character)

    MOV     DH,BL

    POP     BX

    RET

```

```
L_SHOW_CURSOR:           ;Show cursor at current position
```

```

MOV     AL,DRAW$TEXT      ;Send Draw Text Command
CALL    L_PULSE$WR
MOV     AL,1              ;Send 1 character only
CALL    L_PULSE$WR
MOV     AL,'_'
CALL    L_PULSE$WR        ;Send Ascii
MOV     AL,0
CALL    L_PULSE$WR        ;Send Ascii X2 (So we have an even number of bytes sent)
RET

```

```

L_HIDE_CURSOR:           ;Hide cursor at current position

```

```

PUSH    BX
CALL    L_GET_COLOR      ;Normally white on black
PUSH    BX               ;Save For below
MOV     BX,L_BLACK_COLOR
CALL    L_SET_COLOR
MOV     AL,DRAW$TEXT      ;Send Draw Text Command
CALL    L_PULSE$WR
MOV     AL,1              ;Send 1 character only
CALL    L_PULSE$WR
MOV     AL,'_'
CALL    L_PULSE$WR        ;Send Ascii
MOV     AL,0

```

```

CALL    L_PULSE$WR          ;Send Ascii X2 (So we have an even number of bytes sent)
POP     BX                  ;Get Back original color
CALL    L_SET_COLOR
POP     BX
RET

```

L_DO_CR:

```

PUSH    BX                  ;Move cursor to start of line
CALL    L_HIDE_CURSOR
XOR     BX,BX
CALL    L_SET_X
CALL    L_SHOW_CURSOR
POP     BX
RET

```

L_DO_LF:

```

PUSH    BX                  ;Move cursor down vertically one line
CALL    L_HIDE_CURSOR
CALL    L_GET_Y
CMP     BX, (L_SCREEN_LINES * L_CHAR_HEIGHT) - L_CHAR_HEIGHT
JL      L_DO_LF1
CALL    L_SCROLL_UP_1       ;Scroll up 1 line

```

```

        JMP     L_DO_LF2

L_DO_LF1:
        ADD     BX,L_CHAR_HEIGHT
        CALL    L_SET_Y

L_DO_LF2:
        CALL    L_SHOW_CURSOR
        POP     BX
        RET

L_GET_COLOR:                                ;READ Register command, Get Text Color. Data in BX
        MOV     AL,READ$CSR
        CALL    L_PULSE$WR                ;Send
        MOV     AL,0
        CALL    L_PULSE$WR
        MOV     AL,0
        CALL    L_PULSE$WR
        MOV     AL,0H                    ;Point to CSR_COLOR
        CALL    L_PULSE$WR
        CALL    L_PULSE$2RD                ;<<< Read 2 byte into [BX]
        RET

L_SET_COLOR:                                ;WRITE Register command, Set Text Color. Data in BX
        MOV     AL,WRITE$CSR

```

```

CALL    L_PULSE$WR          ;Send
MOV     AL,0
CALL    L_PULSE$WR
MOV     AL,0
CALL    L_PULSE$WR
MOV     AL,0H                ;Point to CSR_COLOR
CALL    L_PULSE$WR
MOV     AL,BH
CALL    L_PULSE$WR
MOV     AL,BL
CALL    L_PULSE$WR
RET

```

```

L_NEXT_POSITION:                ;Advance LAVA cursor to next position (Cursor is not displayed here)
    PUSH    BX
    CALL    L_GET_X              ;Get X position in BX
    CMP     BX,(L_CHARS_PER_LINE * L_CHAR_WIDTH) - L_CHAR_WIDTH
    JL      L_SAME_LINE          ;On Same line just update
    CALL    L_DO_LF
    CALL    L_HIDE_CURSOR
    XOR     BX,BX                ;X=0 always to start of next line
    CALL    L_SET_X
    POP     BX
    RET

```



```
L_SAME_LINE:                ;This is the normal situation
```

```
    ADD     BX,8
```

```
    CALL    L_SET_X
```

```
    POP     BX
```

```
    RET
```

```
L_CLEAR_CURRENT_LINE:        ;Clear a whole line at current cursor Y position (Any X position on that line)
```

```
    PUSH    BX
```

```
    PUSH    CX                ;No need to hide cursor
```

```
    PUSH    SI
```

```
    CALL    L_GET_X
```

```
    PUSH    BX                ;Store it for when we return below
```

```
    CALL    L_GET_Y
```

```
    MOV     SI,BX
```

```
    CALL    L_CLEAR_LINE
```

```
    POP     BX                ;Get Back original Cursor position
```

```
    CALL    L_SET_X
```

```
    POP     SI
```

```
    POP     CX
```

```
    POP     BX
```

```
    RET
```

```
L_CLEAR_SCREEN:                ;Clear the whole screen. Cursor to 0,0
```

```
    PUSH    BX
```

```
    PUSH    CX
```

```
    PUSH    SI
```

```
    MOV     SI,0
```

```
    MOV     CX,L_SCREEN_LINES+1 ;Count of total lines on screen
```

```
L_CLEAR1:
```

```
    PUSH    CX
```

```
    PUSH    SI
```

```
    CALL    L_CLEAR_LINE
```

```
    POP     SI
```

```
    POP     CX
```

```
    ADD     SI,L_CHAR_HEIGHT
```

```
    LOOP    L_CLEAR1
```

```
    XOR     BX,BX                ;Set cursor position 0,0
```

```
    CALL    L_SET_X
```

```
    CALL    L_SET_Y
```

```
    POP     SI
```

```
    POP     CX
```

```
    POP     BX
```

```
    RET
```

```
L_CLEAR_LINE:                ;Clear line at [SI]. Note BX & DX changed
```

```

        MOV     CX,0                ;Count for lines below

L_CLEAR_LINE1:

        MOV     BX,L_BELOW_SCREEN    ;SOURCE: Below bottom of screen (will display as background, see L_CLEAR_BUFFER)
        ADD     BX,CX

        MOV     DX,0                ;X position is 0
        CALL    L_MAKE_24_ADDRESS     ;Generate LAVA 24 bit address DX+BX -> DX+BH


        MOV     AL,COPY$MEMORY        ;Sent COPY MEMORY command
        CALL    L_PULSE$WR

        MOV     AL,BH
        CALL    L_PULSE$WR            ;Source Address 23:16

        MOV     AL,DH
        CALL    L_PULSE$WR            ;Source Address 15:8

        MOV     AL,DL
        CALL    L_PULSE$WR            ;Source Address 7:0


        MOV     AL,03H
        CALL    L_PULSE$WR            ;Next two size bytes

        MOV     AL,0FFH
        CALL    L_PULSE$WR


        MOV     AX,0
        CALL    L_PULSE$WR            ;Filler byte


        MOV     BX,SI                ;DESTINATION:  Get Y position

```

```

ADD    BX,CX
MOV    DX,0                ;X position is 0
CALL   L_MAKE_24_ADDRESS    ;Generate LAVA 24 bit address DX+BX -> DX+BH
MOV    AL,BH
CALL   L_PULSE$WR          ;Send Address 23:16
MOV    AL,DH
CALL   L_PULSE$WR          ;Send Address 15:8
MOV    AL,DL
CALL   L_PULSE$WR          ;Send Address 7:0
INC    CX
CMP    CX,L_CHAR_HEIGHT
JNZ    L_CLEAR_LINE1
RET

```

```

L_CLEAR_EOL:                ;Clear to EOL (Any X position to end of that line)
PUSH   BX
PUSH   CX
PUSH   DX
PUSH   SI
PUSH   DI
CALL   L_HIDE_CURSOR
CALL   L_GET_X              ;GET X POSITION OF CURSOR IN BX (Number of character positions)
PUSH   BX                  ;Save Cursor for when done

```

```

MOV     SI,BX                ;>>> X Position in SI <<<

MOV     DX,L_CRT_WIDTH

SUB     DX,BX                ;>>> Length in DX <<<< of line in delete area left to EOL


CALL    L_GET_Y              ;GET Y POSITION OF CURSOR IN BX (Number of character positions)

MOV     DI,BX                ;>>> Y Position in DI <<<


MOV     CX,0                 ;Count for lines for below

EOL_LINE1:                   ;Will move a clear memory block into the area

PUSH    DX                   ;Save data

MOV     BX,L_BELOW_SCREEN    ;SOURCE: Below bottom of screen (will display as background, see L_CLEAR_BUFFER)

ADD     BX,CX

MOV     DX,SI

CALL    L_MAKE_24_ADDRESS    ;Generate LAVA 24 bit address DX+BX -> DX+BH


MOV     AL,COPY$MEMORY       ;Sent COPY MEMORY command

CALL    L_PULSE$WR

MOV     AL,BH

CALL    L_PULSE$WR           ;Source Address 23:16

MOV     AL,DH

CALL    L_PULSE$WR           ;Source Address 15:8

MOV     AL,DL

CALL    L_PULSE$WR           ;Source Address 7:0


POP     DX                   ;Get back saved length

```

```

MOV     AL,DH

CALL    L_PULSE$WR           ;Next two size bytes

MOV     AL,DL

CALL    L_PULSE$WR

PUSH    DX                   ;Save length again


MOV     AX,0

CALL    L_PULSE$WR           ;Filler byte


MOV     BX,DI                 ;DESTINATION:  Get Y position

ADD     BX,CX

MOV     DX,SI                 ;X position is 0

CALL    L_MAKE_24_ADDRESS     ;Generate LAVA 24 bit address DX+BX -> DX+BH

MOV     AL,BH

CALL    L_PULSE$WR           ;Send Address 23:16

MOV     AL,DH

CALL    L_PULSE$WR           ;Send Address 15:8

MOV     AL,DL

CALL    L_PULSE$WR           ;Send Address 7:0


POP     DX                   ;balance up stack

INC     CX

CMP     CX,L_CHAR_HEIGHT

JNZ     EOL_LINE1

```

```

POP     BX                ;Get Back original Cursor position
CALL    L_SET_X
CALL    L_SHOW_CURSOR
POP     DI
POP     SI
POP     DX
POP     CX
POP     BX
RET

```

```

L_SCROLL_UP_1:            ;Move the whole screen up one line (quickly)
    PUSH    BX
    PUSH    CX
    PUSH    SI
    PUSH    DI
    CALL    L_GET_Y
    PUSH    BX                ;Store it for when we return below
    CALL    L_HIDE_CURSOR

    MOV     CX, (L_CHAR_HEIGHT * L_SCREEN_LINES) - L_CHAR_HEIGHT ;(Count for screen scan lines below)
    MOV     SI, L_CHAR_HEIGHT                ;Source, one line down
    MOV     DI, 0                ;Destination, top of screen

```

```

UP1:  MOV    BX,SI                ;SOURCE

      MOV    DX,0                ;X position is 0

      CALL   L_MAKE_24_ADDRESS    ;Generate LAVA 24 bit address DX+BX -> DX+BH


      MOV    AL,COPY$MEMORY      ;Sent COPY MEMORY command

      CALL   L_PULSE$WR

      MOV    AL,BH

      CALL   L_PULSE$WR          ;Source Address 23:16

      MOV    AL,DH

      CALL   L_PULSE$WR          ;Source Address 15:8

      MOV    AL,DL

      CALL   L_PULSE$WR          ;Source Address 7:0


      MOV    AL,03H

      CALL   L_PULSE$WR          ;Next two size bytes

      MOV    AL,0FFH

      CALL   L_PULSE$WR


      MOV    AX,0

      CALL   L_PULSE$WR          ;Filler byte


      MOV    BX,DI                ;DESTINATION

      MOV    DX,0                ;X position is 0

      CALL   L_MAKE_24_ADDRESS    ;Generate LAVA 24 bit address DX+BX -> DX+BH

      MOV    AL,BH

```



```
CALL    L_PULSE$WR          ;Send Address 23:16
```

```
MOV     AL,DH
```

```
CALL    L_PULSE$WR          ;Send Address 15:8
```

```
MOV     AL,DL
```

```
CALL    L_PULSE$WR          ;Send Address 7:0
```

```
INC     SI                   ;Next scan line
```

```
INC     DI
```

```
LOOP    UP1
```

```
MOV     SI, (L_CHAR_HEIGHT * L_SCREEN_LINES)
```

```
CALL    L_CLEAR_CURRENT_LINE
```

```
POP     BX                   ;Get Back original Y Cursor position
```

```
CALL    L_SHOW_CURSOR
```

```
POP     DI
```

```
POP     SI
```

```
POP     CX
```

```
POP     BX
```

```
RET
```

```
L_DEL_SPACE:                ;DEL requires special treatment because LAVA does not
```

```
    PUSH    BX                ;Overwrite characters. ie a space will not delete a character if overwritten
```

```

    PUSH    CX

    PUSH    DX

    CALL    L_HIDE_CURSOR

    CALL    L_GET_X

    JMP     L_PUT_SPACE

L_BACK_SPACE:                ;Back space requires special treatment because LAVA does not
                               ;Overwrite characters. ie a space will not delete a character if overwritten
    PUSH    BX
    PUSH    CX
    PUSH    DX
    CALL    L_HIDE_CURSOR
    CALL    L_GET_X
    CMP     BX,8
    JL      L_PUT_SPACE
    SUB     BX,8
    CALL    L_SET_X            ;Back space one character

L_PUT_SPACE:                 ;Print a space on Screen at current X,Y position
    CALL    L_GET_COLOR
    PUSH    BX                ;Save for below
    MOV     BX,L_BLACK_COLOR  ;The trick is to overlay with two ASCII characters that fill the complete
    CALL    L_SET_COLOR        ;8X16 pixel area in black (If a different background color, then change)

    MOV     AL,DRAW$TEXT      ;Send Draw Text Command
    CALL    L_PULSE$WR
    MOV     AL,1              ;Send 1 character only
    CALL    L_PULSE$WR

```

```

MOV     AL,03H

CALL    L_PULSE$WR           ;Send Ascii 'Heart figure"

MOV     AL,0

CALL    L_PULSE$WR           ;send Ascii X2 (So we have an even number of bytes sent)


MOV     AL,DRAW$TEXT         ;Send Draw Text Command

CALL    L_PULSE$WR

MOV     AL,1                 ;Send 1 character only

CALL    L_PULSE$WR

MOV     AL,08H

CALL    L_PULSE$WR           ;Overlay with ASCII "Circle figure"

MOV     AL,0

CALL    L_PULSE$WR           ;send Ascii X2 (So we have an even number of bytes sent)


POP     BX                   ;Get back color

CALL    L_SET_COLOR

CALL    L_SHOW_CURSOR

POP     DX

POP     CX

POP     BX

MOV     AL,BS                 ;Return with space in AL

RET

```

```

L_MAKE_24_ADDRESS:           ;Generate LAVA 24 bit memory address from X in DX, and Y in BX

```

```

MOV     AX,DX                ;Get X address, isolate bits 8 & 9
AND     AH,03H
SHL     BX,1                 ;Shift over Y by 2 bits
SHL     BX,1
OR      AH,BL                ;Combine in lower 6 bits of Y coordinate
MOV     DH,AH
RET                                ;Return Address 7:0 in DL, 15:8 in DH, and 23:16 in BH

```

L_GET_X:

```

MOV     AL,READ$CSR          ;READ Register command for X Position into BX
CALL    L_PULSE$WR           ;Send
MOV     AL,0
CALL    L_PULSE$WR
MOV     AL,0
CALL    L_PULSE$WR
MOV     AL,01H               ;Point to CSR_FONT_X
CALL    L_PULSE$WR
CALL    L_PULSE$2RD          ;<<< Read 2 byte into [BX]
RET

```

L_GET_Y:

```

MOV     AL,READ$CSR          ;READ Register command for Y Position into BX
CALL    L_PULSE$WR           ;Send

```

```

MOV     AL,0
CALL    L_PULSE$WR
MOV     AL,0
CALL    L_PULSE$WR
MOV     AL,02H                ;Point to CSR_FONT_Y
CALL    L_PULSE$WR
CALL    L_PULSE$2RD          ;<<< Read 2 byte into [BX]
RET

```

L_SET_X:

```

MOV     AL,WRITE$CSR          ;WRITE Register command, X Position, data in BX
CALL    L_PULSE$WR           ;Send
MOV     AL,0
CALL    L_PULSE$WR
MOV     AL,0
CALL    L_PULSE$WR
MOV     AL,01H                ;Point to CSR_FONT_X
CALL    L_PULSE$WR
MOV     AL,BH
CALL    L_PULSE$WR
MOV     AL,BL
CALL    L_PULSE$WR
RET

```

```

L_SET_Y:
    MOV     AL,WRITE$CSR      ;WRITE Register command, Y Position, data in BX
    CALL    L_PULSE$WR       ;Send
    MOV     AL,0
    CALL    L_PULSE$WR
    MOV     AL,0
    CALL    L_PULSE$WR
    MOV     AL,02H           ;Point to CSR_FONT_Y
    CALL    L_PULSE$WR
    MOV     AL,BH
    CALL    L_PULSE$WR
    MOV     AL,BL
    CALL    L_PULSE$WR
    RET

;Clear an area of the LAVA screen RAM for use with Clear Line, EOL etc.

L_CLEAR_BUFFER:              ;We will us this for fast LAVA block moves etc.
    MOV     BX,L_BELOW_SCREEN ;Y position of RAM below bottom of visible screen
CLEARB2:MOV     DX,0          ;X position is 0
CLEARB1:PUSH    BX
    PUSH    DX
    CALL    L_MAKE_24_ADDRESS ;Generate LAVA 24 bit address DX+BX -> DX+BH

```

```

MOV     AL,WRITE$MEMORY           ;Sent COPY MEMORY command
CALL    L_PULSE$WR
MOV     AL,BH                     ;Address 23:16
CALL    L_PULSE$WR
MOV     AL,DH                     ;Address 15:8
CALL    L_PULSE$WR
MOV     AL,DL                     ;Address 7:0
CALL    L_PULSE$WR
MOV     AL,L_BLACK_COLOR          ;BLACK (0FH = Blue for testing)
CALL    L_PULSE$WR
MOV     AL,L_BLACK_COLOR          ;BLACK
CALL    L_PULSE$WR
POP     DX
POP     BX
INC     DX                        ;Are we at end of line
CMP     DX,L_CRT_WIDTH+10
JLE     CLEARB1
INC     BX                        ;Go to next scan line
CMP     BX,L_BELOW_SCREEN+L_CHAR_HEIGHT ;16 scan lines total
JLE     CLEARB2
RET

```

```

;----- LAVA CORE WRITE ROUTINE -----

```

```

                                ;Note only [AL] altered

L_PULSE$WR:                    ;>>>> WRITE ONE BYTE OF DATA TO LAVA CHIP, Data in [AL] <<<<

    OUT    LavaData,AL          ;Send [AL] to Lava data port (91H)

    MOV    AL,00000001B         ;Output enable U10 to LAVA data bus, and set LAVA to WRITE mode

    OUT    LavaStatus,AL        ;Send to lava status port (90H)

    MOV    AL,10000001B         ;Then pulse status port strobe bit LOW (Bit 7 high, pulsed strobe low)

    OUT    LavaStatus,AL        ;Send to lava status port (90H)

L_WR$NOT$RDY:

    IN     AL,LavaStatus        ;Wait until LAVA "Done" signal clears U12A. Then we are done

    AND    AL,80H               ;This will set strobe bit back HIGH. Note still in WRITE LAVA mode

    JZ     L_WR$NOT$RDY

    RET

;----- LAVA CORE READ ROUTINE -----

                                ;Note only [AL] & [BX] altered

L_PULSE$2RD:                    ;>>>> READ TWO BYTES OF DATA FROM LAVA CHIP, Data in [BX] <<<<

    MOV    AL,00001000B         ;Set to LAVA READ MODE, Disable U10 to LAVA data bus

    OUT    LavaStatus,AL        ;Send to lava status port (91H)

    MOV    AL,10001000B         ;Then pulse strobe bit LOW

    OUT    LavaStatus,AL        ;Send to lava status port (91H)

L_RD$NOT$RDY:

    IN     AL,LavaStatus        ;Wait until JAVA "Done" signal clears U12A. Then we are done

    AND    AL,80H               ;This will set strobe bit back HIGH. Note still in READ LAVA mode

    JZ     L_RD$NOT$RDY

```



```

IN      AL,LavaData      ;Data [15:8] from port (90H)
MOV     BH,AL            ;Save in BH

MOV     AL,10001000B     ;Pulse strobe bit LOW
OUT     LavaStatus,AL    ;Send to lava status port (91H)

```

```
L_RD$NOT$RDY1:
```

```

IN      AL,LavaStatus    ;Wait until JAVA "Done" signal clears U12A. Then we are done
AND     AL,80H           ;This will set strobe bit back HIGH. Note still in READ LAVA mode
JZ      L_RD$NOT$RDY1

                                ;Now Second Byte

IN      AL,LavaData      ;Data [7:0] from port (90H)
MOV     BL,AL

RET                                           ;Return with data in [BX]

```

```
;*****
```

```

;

;      Console Input Handler   (Software Interrupt 16H)
;      Return with keyboard buffer character in AL
;

```

```
;Input:      AH = 00h      KEYBOARD - GET KEYSTROKE
```

```
;
```

```

;Return:AH = BIOS scan code

;      AL = ASCII character

;      Note:  On extended keyboards, this function discards any extended keystrokes,
;              returning only when a non-extended keystroke is available. The BIOS
;              scan code is usually, but not always, the same as the hardware scan
;              code processed by INT 09. It is the same for ASCII keystrokes and most
;              unshifted special keys (F-keys, arrow keys, etc.), but differs for shifted
;              special keys. Some (older) clone BIOSes do not discard extended keystrokes
;              and manage function AH=00h and AH=10h the same.
;
;Input:      AH = 01h      KEYBOARD - CHECK FOR KEYSTROKE
;
;Return:ZF set if no keystroke available
;      ZF clear if keystroke available
;      AH = BIOS scan code
;      AL = ASCII character
;
;      Note: If a keystroke is present, it is not removed from the keyboard buffer;
;              however, any extended keystrokes which are not compatible with 83/84- key keyboards
;              are removed by IBM and most fully-compatible BIOSes in the process of checking
;              whether a non-extended keystroke is available. Some (older) clone BIOSes do not
;              discard extended keystrokes and manage function AH=00h and AH=10h the same.
;
;Input:      AH = 02h      KEYBOARD - GET SHIFT FLAGS
;
;Return:AL = shift flags (see below)

```

```

;      AH destroyed by many BIOSes

;      Bitfields for keyboard shift flags:-

;      7      Insert active
;      6      CapsLock active
;      5      NumLock active
;      4      ScrollLock active
;      3      Alt key pressed (either Alt on 101/102-key keyboards)
;      2      Ctrl key pressed (either Ctrl on 101/102-key keyboards)
;      1      left shift key pressed
;      0      right shift key pressed

;*****

CONIN:  sti
        push    ds
        push    bx
        XOR     BX,BX                ;Set DS to data area for ROM usage in low RAM @ 400H....)
        MOV     DS,BX

Xconio: or      ah,ah                ;read keyboard?
        jnz     conio1              ;skip if not

conio0: TEST     byte[KB_FLAG_1],HOLD_STATE ;Is Pause flag set
        JNZ     conio0

```

```

mov     al,[chrcnt]          ;any data in buffer?
test    al,al
je       con10               ;wait for a key
mov     bx,[bufhd]           ;get buffer address
mov     al,[bx]              ;character to al
mov     ah,0                 ;scan code always zero
inc     bx
cmp     bx,keybuff+32        ;at end of buffer?
jl      con100
mov     bx,keybuff           ;reset buffer address if so
con100: mov    [bufhd],bx
cli                                           ;turn off interrupts
dec     byte [chrcnt]         ;while we adjust count
sti
pop     bx
pop     ds
iret                                ;return char in AL, AH=0

con11:  cmp     ah,1           ;read status?
jne     con12               ;skip if not
mov     al,[chrcnt]         ;get character count
test    al,al               ;Z-flag = availability
mov     bx,[bufhd]
mov     al,[bx]             ;character to al
mov     ah,0                ;scan code = 0

```

```

conirt: pop     bx
        pop     ds
        retf    2                ;throw away flags

conir2: cmp     ah,3              ;read shift status
        jne     conir3
        mov     al,0             ;set status to zero
conir3: pop     bx
        pop     ds
        iret

```

```

;*****

```

```

;
;      Printer Output Handler      (Software Interrupt 17H)
;

```

```

;Input:      AH = 00h      PRINTER - WRITE CHARACTER
;
;      AL = character to write
;
;      DX = printer number (00h-02h)
;

```

```

;Return:AH = printer status

```

```

;
;      Bitfields set for printer status:
;
;      7      not busy
;
;      6      acknowledge

```

```

;          5      out of paper
;          4      selected
;          3      I/O error
;          2-1    unused
;          0      timeout
;
;Input:      AH = 01h      PRINTER - INITIALIZE PORT
;          DX = printer number (00h-02h)
;
;Return:AH = printer status (same as above)
;
;Input:      AH = 02h      PRINTER - GET STATUS
;          DX = printer number (00h-02h)
;
;Return:AH = printer status (see above)
;
;*****

```

```

LST_OUT:PUSH AX                      ;Note we will assume only one printer
        PUSH  BX
        PUSH  CX
        CMP   AH,0                  ;AH=0 Print Character
        JZ    PRN_CHAR
        CMP   AH,1
        JZ    INIT_PRN              ;AH=1 Initilize Printer (Set Font etc)

```

```

        JMP     STATUS_PRN          ;AH=2 Get status

PRN_CHAR:

        CALL    LIST_OUT1          ;AH = 0; Print a character (in AL) on printer

LDONE:  POP     CX

        POP     BX

        POP     AX

        XOR     AH,AH              ;Retur Z  set (and AH = 0 ) if all OK

        IRET                       ;<- Note IRET

STATUS_PRN:                          ;Get List Status

        CALL    LIST_STATUS        ;Must be initilize or a status check. Same ending

        JZ      LDONE              ;If it matches xxxx0110B we are OK

PSTAT:  TEST    AL,00001000B        ;Test for paper out

        JNZ     PAPER_OUT

        POP     CX                  ;Else just return busy signal

        POP     BX

        POP     AX                  ;Just in case return with character in AL

        MOV     AH,00000001B        ;return with timeout bit set

        IRET

PAPER_OUT:

        POP     CX

        POP     BX

        POP     AX                  ;Just in case return with character in AL

        MOV     AH,00100000B        ;Flag for paper out

```

```
IRET
```

```
INIT_PRN:
```

```
    MOV    BX,PRN_INIT_STR        ;Set Font etc.
```

```
INIT_PRN1:
```

```
    MOV    CL,[CS:BX]
```

```
    INC    BX
```

```
    OR     CL,CL
```

```
    JZ     LDONE
```

```
    CALL   LIST_OUT
```

```
    JMP    INIT_PRN1
```

```
LIST_OUT1:                ;Remember can be called by IBM BIOS section or the monitor section
```

```
    MOV    CL,AL            ;For BIOS interrupt printing character is in AL
```

```
LIST_OUT:                ;Within this monitor character is in CL
```

```
    MOV    CH,0FFH         ;Check status up to 255 times
```

```
LO2:  CALL   LIST_STATUS   ;XXXX0110 if ready
```

```
    JZ     LIST_OK
```

```
    DEC    CH
```

```
    JNZ    LO2
```

```
LIST_OK:MOV    AL,0FFH      ;Setup strobe high to low then high
```

```
    OUT    PRINTER_STROBE,AL
```

```
    MOV    AL,CL
```

```
    OUT    PRINTER_OUT,AL   ;Now Data
```

```
    MOV    AL,0FEH         ;Bit 0, STROBE FOR CENTRONICS
```



```

OUT    PRINTER_STROBE,AL
MOV    AL,0FFH                ;Raise strobe again
OUT    PRINTER_STROBE,AL
RET

```

```

LIST_STATUS:                ;Remember can be called by IBM BIOS section or the monitor section
    IN    AL,PRINTER_STATUS
CENSTAT:AND    AL,00001111B    ;XXXX0110 IS READY (BIT 3=PAPER BIT 2=FAULT
    CMP    AL,00000110B        ;BIT 1=SELECT  BIT 0=BUSY
RET

```

```

;*****

```

```

;

```

```

;    BASIC Handler    (Software Interrupt 18h)

```

```

;

```

```

;*****

```

```

basic: PUSH    AX

```

```

    PUSH    BX

```

```

    PUSH    CX

```

```

    MOV     BX,NO_BASIC_MSG    ;Announce we got an BASIC Interrupt

```

```

NO_INT_SUPPORT:                ;Common for warning about un-implemented int

```

```

CALL    PRINT_STRING      ;Send msg pointed to by CS:BX

POP     CX                ;Note this routine is also used by the MS-DOS BIOS section

POP     BX

POP     AX

iret                    ;Remember IRET collects the saved Flags

```

```

;*****

```

```

;

```

```

;      Equipment Check Handler          (Software Interrupt 11H)

```

```

;

```

```

;*****

```

```

equip:  push    ds                ;save data segment

        XOR     AX,AX            ;Set DS to data area for ROM usage in low RAM @ 400H....)

        MOV     DS,AX

        mov     ax,[EQFLAG]

        pop     ds

        iret

```

```

;*****

```

```

;

```

```

;      Memory Size Handler      (Software Interrupt 12H)

```

```
;      BIOS - GET MEMORY SIZE

;      Return:AX = kilobytes of contiguous memory starting at absolute address 00000h

;                  Note: This call returns the contents of the word at 0040h:0013h;

;                  in PC and XT, this value is set from the switches on the motherboard

;*****
```

```
memsiz: push    ds

        XOR     AX,AX                ;Set DS to data area for ROM usage in low RAM @ 400H....)

        MOV     DS,AX

        mov     ax,[memrsz]

        pop     ds

        iret
```

```
;*****

;

;      Interrupt 1Bh Keyboard Break

;

;*****
```

```
kbd_break:

        PUSH    AX

        PUSH    BX

        PUSH    CX
```

```

MOV     BX,NO_BREAK_MSG           ;Announce we got an BREAK Interrupt
      jmp     NO_INT_SUPPORT

```

```

;*****

```

```

;

```

```

;      Interrupt 1Ch (28 Decimal)  User Timer Tic

```

```

;

```

```

;*****

```

```

user_timer:

```

```

      IRET                        ;Just return

```

```

;*****

```

```

;

```

```

;      Comm I/O Handler           (Software Interrupt 14H)

```

```

;

```

```

;      Note: We will leave it at 19,200 Baud (faster than on original PC)

```

```

;

```

```

;Input:      AH = 00h      SERIAL - INITIALIZE PORT

```

```

;      AL = port parameters

```

```

;          Paramater Bit Description
;          7-5    data rate (110,150,300,600,1200,2400,4800,9600 bps)
;          4-3    parity (00 or 10 = none, 01 = odd, 11 = even)
;          2      stop bits (set = 2, clear = 1)
;          1-0    data bits (00 = 5, 01 = 6, 10 = 7, 11 = 8)
;          DX = port number (00h-03h)
;Return:AH = line status

;          Bit(s)  Description
;          7       carrier detect
;          6       ring indicator
;          5       data set ready
;          4       clear to send
;          3       delta carrier detect
;          2       trailing edge of ring indicator
;          1       delta data set ready
;          0       delta clear to send
;
;Input:      AH = 01h      SERIAL - WRITE CHARACTER TO PORT
;          AL = character to write
;          DX = port number (00h-03h)
;Return:AH bit 7 clear if successful
;          AH bit 7 set on error
;          AH bits 6-0 = port status
;
;Input:      AH = 02h      SERIAL - READ CHARACTER FROM PORT

```

```
;      AL = 00h (ArtiCom)

;      DX = port number (00h-03h

;      Return:AH = line status

;      AL = received character if AH bit 7 clear
```

```
;S100Computers Serial I/O Board Initilization
```

```
;Note only SSC A of the Zilog SCC serial ports will be set used (and set to 38,400 Baud initially).
```

```
;Will leave SSC B at 38,400 for speech synthizer (untouched). So DX will be ignored
```

```
commio:      PUSH    AX                      ;Note we will assume only two serial ports
```

```
      PUSH    BX                      ;so DX = 0 or 1
```

```
      PUSH    CX
```

```
      CMP     AH,0
```

```
      JZ      INIT_SIO                ;Initilize serial port
```

```
      CMP     AH,1
```

```
      JZ      WR_SIO                  ;Write to serial port
```

```
      JMP     RD_SIO                  ;Must be AH=2, read from serial port
```

```
INIT_SIO:                      ;Program Channel A
```

```
      MOV     AH,AL                  ;Store Baud etc in AH
```

```
      CMP     DX,0
```

```
      JNZ     SIO_DONE                ;Skip serial ports 1,2 & 3
```

```
      MOV     AL,04H                 ;Point to WR4
```

```

OUT    ACTL,AL
MOV    AL,44H          ;X16 clock,1 Stop,NP
OUT    ACTL,AL

MOV    AL,03H          ;Point to WR3
OUT    ACTL,AL
MOV    AL,0C1H          ;Enable reciever, Auto Enable, Recieve 8 bits
; MOV    AL,0E1H          ;Enable reciever, No Auto Enable, Recieve 8 bits (for CTS bit)
OUT    ACTL,AL

MOV    AL,05H          ;Point to WR5
OUT    ACTL,AL
MOV    AL,0EAH          ;Enable, Transmit 8 bits
OUT    ACTL,AL          ;Set RTS,DTR, Enable

MOV    AL,0BH          ;Point to WR11
OUT    ACTL,AL
MOV    AL,56H          ;Recieve/transmit clock = BRG
OUT    ACTL,AL

MOV    AL,0CH          ;Point to WR12
OUT    ACTL,AL
; MOV    AL,40H          ;Low Byte 2400 Baud (Note can expand later, AH has Baud rate bits)
; MOV    AL,1EH          ;Low Byte 4800 Baud
; MOV    AL,0EH          ;Low Byte 9600 Baud

```



```

WR_SIO:                                ;Write a character to SSC Channel A

    MOV     AH,AL                      ;Store char in AH

    MOV     CX,256                     ;Will try 256 times, then timeout

WR_SIO1:IN     AL,ACTL                  ;(A0), Is SCC TX Buffer empty

    AND     AL,04H

    JNZ     SENDSER                    ;NZ if ready to recieve character

    LOOP    WR_SIO1

BAD_SER:POP     CX

    POP     BX

    POP     AX

    XOR     AH,AH

    OR      AH,80H                     ;Flag we have a problem

    IRET                                ;Note IRET not RET


SENDER:MOV     AL,AH

    OUT     ADTA,AL                    ;(A2), Send it

    JMP     SIO_DONE


RD_SIO:                                ;Read a character from SSC Channel A

    MOV     CX,256                     ;Will try 256 times, then timeout

RD_SIO1:IN     AL,ACTL                  ;(A0), Is SCC TX Buffer empty

    AND     AL,01H

    JNZ     GETSER                     ;NZ if something there

    LOOP    RD_SIO1

    JMP     BAD_SER

```

```

GETSER:      POP      CX                      ;Get back everything

            POP      BX

            POP      AX

            XOR      AH,AH

            IN       AL,ADTA                  ; (A2), return with data

            IRET                             ;Note IRET not RET


SERIAL_OUT:                      ;Simple write a character to SSC Channel#1 on S100Computers Serial IO Board

            MOV      AH,CL                    ;Store char in AH

            PUSH     CX

            MOV      CX,256                  ;Will try 256 times, then timeout

SERIAL_OUT1:

            IN       AL,ACTL                  ; (A0), Is SCC TX Buffer empty

            AND      AL,04H

            JNZ      SERIAL_OUT2              ;NZ if ready to recieve character

            LOOP     SERIAL_OUT1

            POP      CX

            XOR      AH,AH

            OR       AH,80H                  ;Flag we have a problem

            RET                             ;Note RET not IRET


SERIAL_OUT2:

            MOV      AL,AH

            OUT      ADTA,AL                  ; (A2), Send it

```

```

POP     CX                ;We will assume no problem, always!

XOR     AH,AH             ;Z for no problem

RET                     ;Note RET not IRET

```

```

;*****

```

```

;

```

```

;      Old Cassette Handler          (Software Interrupt 15H)

;      We will use this as a staging point for a far Jump if an extra

;      ROM is discovered during the BIOS initialization sequence

;      Things like SCSI adaptors etc.

```

```

;

```

```

;*****

```

```

CASSETTE:

```

```

    push     DS

```

```

PUSH     AX

```

```

XOR     AX,AX             ;Set DS to data area for ROM usage in low RAM @ 400H....)

```

```

MOV     DS,AX

```

```

CMP     byte [DEBUG_FLAG],0 ;Is Debug mode on

```

```

JNZ     CASSETTE_DEBUG     ;If not 0 then send debug data

```

```

PUSH     AX

```



```
retf    2                ;Remove the original status flags on return
```

```
Cassette2:
```

```
    CMP    AH,41H        ;Extenal Wait event (Unused)
```

```
    JZ     EXT_WAIT
```

```
    CMP    AH,0C0H
```

```
    JZ     GET_DESCRIPTION_TABLE
```

```
    CMP    AH,0C1H        ;RETURN EXTENDED-BIOS DATA-AREA SEGMENT ADDRESS (PS)
```

```
    JZ     EXT_BIOS_DATA
```

```
    CMP    AH,88H
```

```
    JZ     HIGH_RAM_CHECK
```

```
    PUSH   AX
```

```
    PUSH   BX
```

```
    PUSH   CX
```

```
    PUSH   AX
```

```
    MOV    BX,CASSETTE_MSG    ;Announce we got an EXTRA Interrupt
```

```
    CALL   PRINT_STRING      ;Send msg pointed to by CS:BX
```

```
    POP    AX
```

```
    MOV    AL,AH
```

```
    CALL   AL_HEXOUT
```

```
    MOV    BX,H_MSG_CRLF     ;"H",CR,LF
```

```
CALL    PRINT_STRING        ;Send msg pointed to by CS:BX
```

```
POP     CX
```

```
POP     BX
```

```
POP     AX
```

```
STC                                ;Set carry to indicate INT is not supported
```

```
retf    2                          ;Remove the original status flags on return (remember we got here via an INT)
```

```
GET_DESCRIPTION_TABLE:           ;AH=C0H
```

```
MOV     AX,CS                     ;Return pointer with ES:BX
```

```
MOV     ES,AX
```

```
MOV     BX,SYS_TABLE
```

```
XOR     AX,AX
```

```
CLC                                ;Clear carry
```

```
retf    2                          ;Remove the original status flags on return
```

```
HIGH_RAM_CHECK:                 ;AH=88H
```

```
MOV     AX,0h                    ;Using 8086, so 0H RAM above 1M
```

```
CLC                                ;Set carry
```

```
retf    2                          ;Remove the original status flags on return
```

```
EXT_WAIT:                       ;AH=41H
```

```
STC                                ;Set carry
```

```
retf    2                          ;Remove the original status flags on return
```

```

EXT_BIOS_DATA:                ;AH= C1H, Extended BIOS Data Area Segment in ES

    STC                        ;Set carry  (Used on PS/2, not needed here)

    retf    2                  ;Remove the original status flags on return

```

```

;----- SUPPORT ROUTINES FOR IBM-PC BIOS -----

```

```

dumpreg:                      ;Dump all 8086 registers to screen

```

```

    CALL    PRINT_8086_REGISTERS

```

```

    CALL    PRINT_SEG_REGISTERS

```

```

    RET

```

```

SERIAL_DISPLAY_REGISTERS:      ;For Debugging only, Send to serial port Register values of registers with INTs

```

```

    PUSH    AX                ;Save everything

```

```

    PUSH    BX

```

```

    PUSH    CX

```

```

    PUSH    DX

```

```

    PUSH    DX                ;we will display in this order

```

```

    PUSH    CX

```

```

    PUSH    BX

```

```

    PUSH    AX

```

```
MOV     BX,INT_AX_MSG           ;"AX="
CALL    SERIAL_PRINT_STRING
POP     AX
CALL    SERIAL_AX_HEXOUT       ;Get AX

MOV     BX,INT_BX_MSG           ;"H BX="
CALL    SERIAL_PRINT_STRING
POP     AX                     ;Get BX
CALL    SERIAL_AX_HEXOUT

MOV     BX,INT_CX_MSG           ;"H CX="
CALL    SERIAL_PRINT_STRING
POP     AX                     ;Get CX
CALL    SERIAL_AX_HEXOUT

MOV     BX,INT_DX_MSG           ;"H DX="
CALL    SERIAL_PRINT_STRING
POP     AX                     ;Get DX
CALL    SERIAL_AX_HEXOUT

MOV     BX,H_Msg                ;"H"
CALL    SERIAL_PRINT_STRING

POP     DX                     ;Restore everything
POP     CX
```



```
POP    BX
```

```
POP    AX
```

```
RET
```

```
;=====CORE  SUPPORT ROUTINES =====
```

```
;      Calculate length difference between DS:SI(end) and ES:DI(start)
```

```
CLENGTH:
```

```
MOV    AX,DS          ;DS has segment of final value
MOV    CX,ES          ;ES has segment of start value
SUB    AX,CX          ;Check if finish is the next segment up
JZ     SAME_SEGMENT
CMP    AX,1000H        ;Max length must be < 64K
JG     BAD_RANGE
MOV    AX,0FFFFH
SUB    AX,DI          ;Calculate start up to end of segment
ADD    AX,SI          ;Add in the part from the next segment up.
INC    AX             ;Count = difference +1
MOV    CX,AX          ;Return value in CX
RET
```

```
SAME_SEGMENT:
```

```
MOV    CX,SI
```

```

sub    CX,DI
CMP    CX,0FFFEH
JZ     BAD_RANGE
inc    cx                ;count = difference +1
ret

```

BAD_RANGE:

```

PUSH    BX
PUSH    CX
MOV     BX,RangeErrMsg    ;Range error
CALL    PRINT_STRING
jmp     ToMonitor          ;Note this will clean up the stack

```

```

; Send to console the address ES+DI    ;CX Unchanged

```

SHOW_ADDRESS_ES:

```

push    cx
mov     ax,es
mov     cl,12
shr     ax,cl            ;Get high nibble down to AL
call    hexdigout
MOV     BX,DI

```

```

call  BX_HEXOUT      ;Then next 4 digits in BX
call  BLANK
pop    cx
ret

```

```

SHOW_ADDRESS_ES_NOSPACE:      ;Same but no trailing blank
    push    cx
    mov     ax,es
    mov     cl,12
    shr     ax,cl          ;Get high nibble down to AL
    call    hexdigout
    MOV     BX,DI
    call    BX_HEXOUT      ;Then next 4 digits in BX
    pop     cx
    ret

```

```

;      BINARY OUTPUT      ;Send what is in [al] in bits
AL_BINOUT:      ;No registers altered (except AL)
    push    cx
    mov     cx,8
binout1: push cx
    shl     al,1
    jb      bout1

```

```

        mov     cl,'0'
        push    ax
        call    CO
        pop     ax
        jmp     binend
bout1:  mov     cl,'1'
        push    ax
        call    CO
        pop     ax
binend:  pop     cx
        loop    binout1
        pop     cx
        ret

```

```

;      HEXCHK                ;check for a valid HEX DIGIT

HEX_check:
        sub     al,'0'                ;convert to binary if ok set carry if problem
        jnb     hret
        cmp     al,0ah
        cmc
        jnb     hret
        sub     al,7
        cmp     al,10
        jnb     hret

```

```

        cmp     al,16
        cmc
hret:   ret

```

```

;  Send to console the address DS+SI      ;CX Unchanged

```

```
SHOW_ADDRESS_DS:
```

```

    push     cx                ;Same but send upper nibble of ds reg
    mov      ax,ds
    mov      cl,12
    shr      ax,cl            ;Get high nibble down to AL
    call     hexdigout
    MOV      BX,SI
    call     BX_HEXOUT        ;Then next 4 digits in BX
    call     BLANK
    pop      cx
    ret

```

```

;  Send to console the address SS+SI      ;Used (Only) by sector display routine. CX Unchanged

```

```
SHOW_ADDRESS_SS:
```

```

    push     cx                ;Same but send upper nibble of ds reg

```

```

mov     ax,ss
mov     cl,12
shr     ax,cl           ;Get high nibble down to AL
call    hexdigout
MOV     BX,SI
call    BX_HEXOUT       ;Then next 4 digits in BX
call    BLANK
pop     cx
ret

```

```

;      Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged), terminator in AH - normally 0

```

```

GET2DIGITS:

```

```

PUSH    BX
PUSH    CX
mov     bx,0           ;Default to 0H

call    CICO           ;1st Console input digit to AL
cmp     al,'0'         ;alphanumeric?
jb      bexit2
call    HEX_check      ;convert to binary and check it
jb      err2
add     bl,al          ;Move into BX
mov     cl,4
shl     bx,cl          ;shift in last addition to high nibble on BL

```

```

push    BX                ;Just in case
call    CICO              ;2nd Console input digit to AL
pop     BX

cmp     al,'0'            ;alphanumeric?
jb      bexit2

call    HEX_check         ;convert to binary and check it
jb      err2

add     bl,al             ;Move into BX
MOV     AL,BL

MOV     AH,0              ;Ret 0 in AH if all OK
POP     CX
POP     BX

ret

err2:   POP     CX        ;Cleanup stack
        POP     BX
        JMP     ERR      ;Then normal error exit

bexit2: cmp     al,' '    ;save terminator, if SP,CR accept only 1 digit
        je      bgood2
        cmp     al,', '
        je      bgood2
        cmp     al,CR

```

```

je      bgood2
cmp     al,ESC
je      bgood2
POP     CX                ;Cleanup stack
POP     BX
JMP     ERR              ;Then normal error exit

```

```

bgood2:  mov     ah,al          ;Save SP,', ' or CR in AH
MOV     BH,0
mov     cl,4
shr     bx,cl              ;shift down last addition to low nibble on BL
MOV     AL,BL
POP     CX
POP     BX
ret

```

```

;      Get (up to) 16 bit value (4 digits) to DI. Termination byte in AH

```

```

GET4DIGITS:
PUSH    BX
PUSH    CX
MOV     CX,5              ;4 characters maximum + CR
mov     bx,0
loop4b: call    CICO        ;Console input to AL

```



```

cmp    al,'0'                ;alphanumeric?
jb     bexit
push   cx
mov     cl,4
shl     bx,cl                ;shift in last addition
pop     cx
call    HEX_check            ;convert [AL] to binary and check it
jb     AddressError
add     bl,al
loop    loop4b
MOV     DI,BX
POP     CX
POP     BX
ret                                ;Will return BX = xxxxH

```

```

;      Get (up to) 20 bit parameter. 16 bit value (4 digits) to DI.

```

```

;      If 5 digits, first digit entered to ES (BX,CX, DX unaltered)

```

```

GET5DIGITS:                    ;Will return ES=000xH, DI = xxxxH

```

```

    PUSH    BX

```

```

    PUSH    CX

```

```

    mov     cx,6              ;Max count of 5 characters + CR

```

```

    mov     bx,0              ;So initially ES=0, see below

```

```

loopb: call    CICO            ;Console input to AL

```

```

    cmp     al,'0'                ;alphanumeric?
    jnb     bexit

    push    cx                    ;Save character count
    push    bx                    ;force the highest nibble to ds:
    and     bx,0f000h
    mov     es,bx
    pop     bx
    mov     cl,4
    shl     bx,cl                 ;shift in last addition
    pop     cx
    call    HEX_check             ;convert to binary and check it
    jnb     AddressError
    add     bl,al
    loop    loopb                 ;Do up to 5 characters

bexit: MOV     DI,BX              ;Move data to DI
    cmp     al,' '                ;Terminate with a SP, ", " or CR only
    je      bgood
    cmp     al,', '
    je      bgood
    cmp     al,CR
    je      bgood
    jmp     ERR

bgood: mov     ah,al              ;Save terminator

```

```

    POP     CX                ;Balance up stack
    POP     BX
    ret

```

AddressError:

```

    MOV     BX,AddressErrMsg  ;Range error
    CALL    PRINT_STRING
    jmp     ToMonitor         ;Note this will clean up the stack

```

```

;      For debugging display

```

DEBUG_AX:

```

    PUSH    AX
    PUSH    BX
    PUSH    CX
    CALL    AX_HEXOUT
    POP     CX
    POP     BX
    POP     AX
    RET

```

```

;      Display ALL 8086 registers

```

```

PRINT_8086_REGISTERS:                ;Print AX,BX,CX,DX,SI & DI Registers

    PUSHF                            ;Will print all on CRT on one line followed by a CRLF

    PUSH    AX

    PUSH    BX

    PUSH    CX

    PUSH    SI                        ;Will pop from stack from here

    PUSH    DI

    PUSH    DX

    PUSH    CX

    PUSH    BX

    PUSH    AX

    MOV     BX,AXMSG                  ; [AX]=

    CALL    PRINT_STRING

    POP     AX

    CALL    AX_HEXOUT

    MOV     BX,BXMSG                  ; [BX]=

    CALL    PRINT_STRING

    POP     AX

    CALL    AX_HEXOUT

    MOV     BX,CXMSG                  ; [CX]=

    CALL    PRINT_STRING

```

```
POP     AX
CALL    AX_HEXOUT

MOV     BX,DXMSG           ; [DX]=
CALL    PRINT_STRING
POP     AX
CALL    AX_HEXOUT

MOV     BX,DIMSG           ; [DI]=
CALL    PRINT_STRING
POP     AX
CALL    AX_HEXOUT

MOV     BX,SIMSG           ; [SI]=
CALL    PRINT_STRING
POP     AX
CALL    AX_HEXOUT

MOV     BX,H_MSG           ;Final H
CALL    PRINT_STRING
POP     CX
POP     BX
POP     AX
POPF
RET
```

```
;   Display 8086 Segment registers
```

```
PRINT_SEG_REGISTERS:                ;Print current RAM loction of the stack

    PUSHF                          ;Will print all on CRT on one line followed by a CRLF

    PUSH    AX

    PUSH    BX

    PUSH    CX

    MOV     BX,SSMSG                ; [SS]=

    CALL    PRINT_STRING

    MOV     AX,SS

    CALL    AX_HEXOUT

    MOV     BX,SPMSG                ; [SP]=

    CALL    PRINT_STRING

    MOV     AX,SP

    SUB     AX,10                   ;Adjust because we saved stuff first

    CALL    AX_HEXOUT

    MOV     BX,CMSG                ; [CS]=

    CALL    PRINT_STRING

    MOV     AX,CS

    CALL    AX_HEXOUT
```

```

MOV     BX,DSMSG           ; [DS]=
CALL    PRINT_STRING
MOV     AX,DS
CALL    AX_HEXOUT

```

```

MOV     BX,ESMSG           ; [ES]=
CALL    PRINT_STRING
MOV     AX,ES
CALL    AX_HEXOUT

```

```

MOV     BX,BPMSG           ; [BP]=
CALL    PRINT_STRING
MOV     AX,BP
CALL    AX_HEXOUT

```

```

MOV     BX,H_MSG           ;Final H
CALL    PRINT_STRING
POP     CX
POP     BX
POP     AX
POPF
RET

```

```

;      CHECK FOR ^S or ESC  AT CONSOL

```

CTRL_CHECK:

```

    call    CSTS
    cmp     al,0
    jz      ctlexit
    call    CICO
    cmp     al,'S'-40h      ;^S will pause
    jnz     ctlcchek       ;possibly ^C

```

xwait: call CSTS

```

    cmp     al,0
    jz      xwait
    ret

```

ctlcchek:

```

    cmp     al,ESC         ;ESC will abort
    jz      ERR

```

ctlexit:ret

; SEND CRLF with an ESC at keyboard check

CRLF_CHECK:

```

    push    cx
    push    bx
    call    CTRL_CHECK      ;Will jump to err if ESC
    mov     cl,CR
    call    CO

```



```
mov    cl,LF
call   CO
pop     bx
pop     cx
ret
```

```
;      SIMPLE SEND CRLF
```

```
CRLF:  push    cx
        push    bx
        mov     cl,CR
        call    CO
        mov     cl,LF
        call    CO
        pop     bx
        pop     cx
        ret
```

```
;      PRINT A BLANK SPACE
```

```
BLANK: push    cx
        mov     cx,1
        call    TABS
        pop     cx
        ret
```

```

;      TABS                                ;[cx] = number of spaces

TABS:  push    cx
        mov     cl,' '
        call    CO
        pop     cx
        loop    TABS
        ret

;      ERROR ABORT ROUTINE

ERR:   MOV     BX,ERR_MSG                    ;Invalid Command (or code not yet done)
        CALL    PRINT_STRING
        jmp     ToMonitor

;      BX_HEXOUT                            ;bx output as 4 hex digits

BX_HEXOUT:                                ;No registers altered
        push    ax
        mov     al,bh
        call    AL_HEXOUT
        mov     al,bl
        call    AL_HEXOUT

```

```
pop    ax
```

```
ret
```

```
;      AX_HEXOUT                ;output the 4 hex digits in [AX]
```

```
AX_HEXOUT:                      ;No registers altered
```

```
    PUSH    AX
```

```
    MOV     AL,AH
```

```
    CALL    AL_HEXOUT
```

```
    POP     AX
```

```
    CALL    AL_HEXOUT
```

```
    RET
```

```
;      AL_HEXOUT                ;output the 2 hex digits in [AL]
```

```
AL_HEXOUT:                      ;No registers altered (except AL)
```

```
    push    cx
```

```
    push    ax
```

```
    mov     cl,4                ;first isolate low nibble
```

```
    shr     al,cl
```

```
    call    hexdigout
```

```
    pop     ax
```

```
    call    hexdigout          ;get upper nibble
```

```
    pop     cx
```

```
ret
```

```
hexdigout:
```

```
    and    al,0fh           ;convert nibble to ascii
    add    al,90h
    daa
    adc    al,40h
    daa
    mov    cl,al
    call   CO
    ret
```

```
;  ROUTINE TO PRINT A STRING  CS:BX = START OF STRING  $ or 0 = FINISH
```

```
PRINT_STRING:
```

```
    push   cx
print1:    mov    al,[CS:bx]      ;Note this routine does NOT assume DS = CS here.
    inc    bx                   ;By using the CS over-ride we will always have
    cmp    al,'$'               ;a valid pointer to messages at the end of this monitor
    jz     print2
    cmp    AL,0                  ;Also terminate with 0's
    JZ     print2
    mov    cl,al
    call   CO
```

```

        jmp    print1
print2:    pop    cx
        ret

;   ROUTINE TO PRINT A STRING  TO S100Computers Serial Port #1  BX = START OF STRING  $ or 0 = FINISH
;   This routine is used mainly for Debugging the IBM BIOS section. No registers altered

SERIAL_PRINT_STRING:
        push    AX
        push    cx
sprint1:mov    al,[CS:bx]        ;Note this routine does NOT assume DS = CS here.
        inc     bx              ;By using the CS over-ride we will always have
        cmp     al,'$'          ;a valid pointer to messages at the end of this monitor
        jz      sprint2
        cmp     AL,0
        JZ      sprint2
        mov     cl,al
        call    SERIAL_OUT      ;Send to serial port #1
        jmp     sprint1
sprint2:pop    cx
        pop     AX
        ret

;   SERIAL_AX_HEXOUT          ;Output the 4 hex digits in [AX] to serial port (used for debugging)

```

```

SERIAL_AX_HEXOUT:                ;No registers altered

    PUSH    AX

    MOV     AL,AH

    CALL    SERIAL_AL_HEXOUT

    POP     AX

    CALL    SERIAL_AL_HEXOUT

    RET


;      SERIAL_AL_HEXOUT          ;output the 2 hex digits in [AL]
SERIAL_AL_HEXOUT:                ;No registers altered (except AL)

    push    cx

    push    ax

    mov     cl,4                  ;first isolate low nibble

    shr     al,cl

    call    SERIAL_hexdigout

    pop     ax

    call    SERIAL_hexdigout      ;get upper nibble

    pop     cx

    ret


SERIAL_hexdigout:

    and     al,0fh                ;convert nibble to ascii

    add     al,90h

    daa

```



```
BELL1: MOV    AL, 06H                ;SEND A BELL
```

```
        OUT    KEYOUT,AL
```

```
        MOV    AL, 1FH
```

```
        CALL   DELAY
```

```
        MOV    AL, CL
```

```
        OUT    KEYOUT,AL
```

```
        RET
```

```
DELAY: DEC    AL                    ;GENERAL COUNT DOWN TIME DELAY
```

```
        JNZ    LX4
```

```
        RET                                ;LENGTH SET IN [A]
```

```
LX4:   PUSH   AX
```

```
        MOV    AL, 05H
```

```
MORE:  DEC    AL
```

```
        PUSH   AX
```

```
        XOR    AL,AL
```

```
MORE2: DEC    AL
```

```
        JNZ    MORE2
```

```
        POP    AX
```

```
        JNZ    MORE
```

```
        POP    AX
```

```
        JMP    DELAY
```


[illegible][illegible]

```
NONE:  XOR    AL,AL
        RET
```

```
;<<<<<<<<<<<<<<< MAIN CONSOL INPUT ROUTINE >>>>>>>>>>>>>>>>
```

```

CI:      CALL    CSTS                ;Wait until something is there

        JZ       CI

        IN       AL,KEYIN

        AND      AL,7FH

        RET

```

```
; <<<<<<<<<<<<< CONSOLE INPUT WITH ECHO ON CONSOLE + LC->UC <<<<<<<<<<
```

```

CICO:  CALL  CI                ;Char -> AL

      AND   AL,7FH

      JZ    BAD_CHAR          ;No Nulls

      CMP   AL,', '           ;Allow "," character

      JZ    CIC1

      CMP   AL,CR              ;ACCEPT ONLY CR,LF,SP

      JZ    CIC1

      CMP   AL,LF

      JZ    CIC1

      CMP   AL,SPACE

      JZ    CIC1

      CMP   AL,ESC             ;Also ESC

      JZ    CIC1


      CMP   AL,'0'

      JB    BAD_CHAR

      CMP   AL,':'             ;Allow 0-9

      JB    CIC1

      CMP   AL,'A'

      JB    BAD_CHAR          ;do not allow : to @

      CMP   AL,'['             ;Is upper case A to Z

      JB    CIC1

      CMP   AL,'a'

      JB    BAD_CHAR

```

```

        CMP     AL,'{'
        JB      UPPER_CASE
        JMP     BAD_CHAR
UPPER_CASE:
        AND     AL,5FH           ;THIS CONVERTS ALL LC->UC
CIC1:  PUSH     AX
        PUSH     CX
        MOV     CL,AL
        CALL    CO               ;DISPLAY ON CONSOLE
        POP     CX
        POP     AX
        RET
;
BAD_CHAR:
        MOV     AL,BELL          ;SEND BELL TO INDICATE BAD DATA
        CALL    CIC1
        MOV     AL,'?'          ;SEND ? TO INDICATE BAD DATA
        JMP     CIC1

SPEAKOUT:
        MOV     AL,0H            ;Will try 256 times, then timeout
SOUT1: PUSH     AX
        IN      AL,BCTL          ;Are we ready for a character
        AND     AL,04H

```

```

        JNZ     SENDS

        POP     AX

        DEC     AL

        JNZ     SOUT1

        RET

SENDS:  POP     AX

        MOV     AL,CL

        OUT     BDTA,AL                ;Send it

        RET

;SPEAKTOMM THIS IS A ROUTINE TO SEND A STRING TO TALKER [BX] AT STRING

SPEAK_STRING:

        MOV     AL,[CS:BX]

        CMP     AL,'$'                ;Terminate with "$" or 0

        JZ      STOMM1

        OR      AL,AL

        JZ      STOMM1

        MOV     CL,AL

        CALL    SPEAKOUT

        INC     BX

        JMP     SPEAK_STRING

STOMM1:  MOV     CL,CR                ;MUST END WITH A CR

        JMP     SPEAKOUT

```

```

POO:  RET                                ;NO PUNCH OUTPUT AT THE MOMENT

RI:   MOV    AL,1AH                      ;NO READER AT THE MOMENT

      RET

```

```

NOT_DONE_WARNING:

```

```

      mov     bx,TO_BE_DONE              ;Signon notice

      call    PRINT_STRING

      RET

```

```

      ;End of the bios code

```

```

;+++++

```

```

;      Data contained in BIOS (Does not get modified, rommable)

```

```

;***** DATA SECTION *****

```

```

;

```

```

;

```

```

;      Interrupt vector table for 8259A

```

```

vec_tbl_8259A:                                ;Pointer to 8259A Hardware interrupts used here

```

```

dw      timer           ;Interrupt Base + 0 ;Will use timer
dw      keyhnd          ;Interrupt Base + 1 ;Will use for keyboard press
dw      Send_EOI        ;Interrupt Base + 2
dw      Send_EOI        ;Interrupt Base + 3
dw      Send_EOI        ;Interrupt Base + 4
dw      Send_EOI        ;Interrupt Base + 5
dw      Send_EOI        ;Interrupt Base + 6
dw      Send_EOI        ;Interrupt Base + 7

```

```

vec_tbl_soft_ints:                ;Pointer to software interrupts used here

```

```

dw      CONOUT          ;interrupt 10
dw      equip           ;interrupt 11
dw      memsiz          ;interrupt 12
dw      DISKIO          ;interrupt 13
dw      commio          ;interrupt 14
dw      CASSETTE        ;interrupt 15
dw      CONIN           ;interrupt 16
dw      LST_OUT         ;interrupt 17
dw      basic           ;interrupt 18
dw      BOOT_DOS_INT    ;interrupt 19
dw      time_of_day     ;interrupt 1A
dw      kbd_break       ;interrupt 1B
dw      user_timer      ;interrupt 1C
dw      VID_PARM_TABLES ;interrupt 1D
dw      FDISK_3PARAM_TBL ;interrupt 1E      ;Default to 3" 1.44M Floppy Disk

```

```
dw      0                ;interrupt 1F
```

```
;Paramaters that MUST be in low RAM for the Lomas CGA board
```

```
INITIAL_VGA_VALUES:
```

```
;to initilize properly with MS-DOS. (Starting at 0:449H...)
```

```
DB      03H              ;CRT_MODE
```

```
DW      0050H            ;CRT_COLS
```

```
DW      1000H            ;CRT_LEN
```

```
DW      0000H            ;CRT_START
```

```
DW      0000H,0000H,0000H,0000H      ;CURSOR_POSN (8 Total) (Must be 0000 for first one at initialization!)
```

```
DW      0000H,0000H,0000H,0000H
```

```
DW      0607H            ;CURSOR_MODE
```

```
DB      0H               ;ACTIVE_PAGE
```

```
DW      3D4H             ;ADDR_6845
```

```
DB      29H              ;CRT_MODE_SET
```

```
DB      30H              ;CRT_PALLETTE
```

```
IVGA_VAL_LEN EQU $-INITIAL_VGA_VALUES
```

```
VID_PARM_TABLES db      38h,28h,2dh,0ah,1fh,6,19h      ;for 40 x 25
```

```
db      1ch,2,7,6,7
```

```
db      0,0,0,0
```

```
db      71h,50h,5ah,0ah,1fh,6,19h      ;for 80 x 25
```

```
db      1ch,2,7,6,7
```

```
db      0,0,0,0
```



```

FDISK_5PARAM_TBL db    0DFH          ;For 5" 360K Disks
                  db     2
                  db    25          ;Time delay for motor
                  db     2          ;512 byte sectors
                  db    09H          ;sectors per track!
                  db    02ah        ;GAP length
                  db    0ffh        ;DTL
                  db    050h        ;GAP length for format
                  db    0f6h        ;Fill byte for format
                  db    25          ;Head settle time
                  db     4          ;Motor stat time
                  db    11          ;length of Table

```

```

FDISK_3PARAM_TBL db    0AFH          ;For 3" 1.44M Disks
                  db     2
                  db    25          ;Time delay for motor
                  db     2          ;512 byte sectors
                  db    12H          ;18 sectors per track
                  db    1BH          ;GAP length
                  db    0FFH        ;DTL
                  db    6CH          ;GAP length for format
                  db    0F6H        ;Fill byte for format
                  db    0FH          ;Head settle time
                  db     8          ;Motor stat time

```

```
db      11          ;length of Table
```

```
;      Default Hard Disk Parameters Table:-
```

```
;      Custom HDISK:  1024 Cylinders, 15 heads, 63 sectors,  512MB Total
```

```
HDISK_PARM_TBL  DW      DOS_MAXCYL      ;0, Max Cylinders

                DB      DOS_MAXHEADS ;2, Max heads (15)

                DW      0000H          ;3, Not used on AT

                DW      0FFFFH         ;5, Start Write Precomp (not used)

                DB      0H              ;7, ECC burst length (not used)

                DB      08H             ;8, "Control Byte" (Bit 7 = disable retrys)

                DB      0H,0H,0H        ;9, Timeouts no used on AT

                DW      0400H           ;A, Landing zone

                DB      DOS_MAXSEC      ;B, Sec/track

                DB      0H,0H,0H,0H     ;C, Reserved


SYS_TABLE       DW      8H              ;Called by INT 15H, AH=C0H called by MSDOS V3+

                DB      0FCH           ;Machine ID Byte

                DB      0               ;Sub model

                DB      0               ;BIOS version

                DB      10H             ;Keyboard Int

                DB      0,0,0
```

```
;      Interrupt messages for checkout
```

```
;
```

```
msg10  db      13,10,'Int 10h',0
```

```
msg11  db      13,10,'Int 11h',0
```

```
msg12  db      13,10,'Int 12h',0
```

```
msg13  db      13,10,'Int 13h',0
```

```
msg14  db      13,10,'Int 14h',0
```

```
msg15  db      13,10,'Int 15h',0
```

```
msg16  db      13,10,'Int 16h',0
```

```
msg17  db      13,10,'Int 17h',0
```

```
msg18  db      13,10,'Int 18h',0
```

```
msg19  db      13,10,'Int 19h',0
```

```
msg1a  db      13,10,'Int 1Ah',0
```

```
msg1b  db      13,10,'Int 1Bh',0
```

```
msg1c  db      13,10,'Int 1Ch',0
```

```
msg1d  db      13,10,'Int 1Dh',0
```

```
msg1e  db      13,10,'Int 1Eh',0
```

```
msg1f  db      13,10,'Int 1Fh',0
```

```
;
```

```
xtmsg  db      13,10,'  Exit',0
```

```
;MAIN MENU COMMAND BRANCH TABLE
```

```

ctable    dw    MAP            ;A            ;Display Memory Map
          dw    SET_CO_FLAG    ;B            ;Set Console output to Propeller or CGA/VGA Video board
          DW    MMENU_FBOOT_DOS ;C            ;LOAD MS-DOS from 5" Floppy ((No debugging))
          dw    DISPLAY_RAM_BYTES ;D          ;Display Memory contents (Read RAM in Bytes)
          dw    SET_TIME_DATE ;E            ;Display/Set Time & Date
          dw    FILL            ;F            ;Fill memory contents
          dw    GOTO            ;G            ;Jump to a SEG:ADDRESS location
          dw    HEXMATH          ;H            ;Add & Subtract two Hex numbers
          dw    SOFT_INTS       ;I            ;Test Software interrupts
          dw    TEST_RAM        ;J            ;Test RAM
          dw    KCMD            ;K            ;Display this menu
          dw    TEST_8259       ;L            ;Test 8259A hardware
          dw    MOVE            ;M            ;Move memory
          dw    MYIDE           ;N            ;Sub-menu to test/diagnose IDE Board
          dw    ERR             ;O            ;
          dw    MMENU_HBOOT_DOS ;P            ;LOAD MS-DOS from HDISK (No debugging)
          dw    QUERY           ;Q            ;Query In or Out to a port
          dw    REGISTERS       ;R            ;Display the 8086 registers in RM
          dw    SUBSTITUTE      ;S            ;Substitute byte values in RAM
          dw    DISPLAY_RAM_WORDS ;T          ;Display Memory contents (Read RAM in Words)
          dw    INPORTS         ;U            ;Display all active 8086 INPUT ports
          dw    VERIFY          ;V            ;Verify two memory regions are the same
          dw    XMODEM_LOAD     ;W            ;Load code to RAM from Modem/Serial port
          dw    IBM_BIOS        ;X            ;IBM-PC Sub menu
          DW    ERR             ;Y            ;

```

```
dw      Z80          ;Z          ;Return back to Z80 master
```

```
        ;IDE COMMAND BRANCH TABLE
```

```
IDE_TABLE  DW  SET_DRIVE_A          ; "A"  Select Drive A
            DW  SET_DRIVE_B          ; "B"  Select Drive B
            DW  ERR                  ; "C"  LOAD CPM (If present)
            DW  DISPLAY              ; "D"  Sector contents display:- ON/OFF
            DW  RAMCLEAR             ; "E"  Clear RAM buffer
            DW  FORMAT               ; "F"  Format current disk
            DW  ERR                  ; "G"  Restore backup
            DW  ERR                  ; "H"  Backup partition
            DW  NEXT_SECT           ; "I"  Next Sector
            DW  PREV_SECT           ; "J"  Previous sector
            DW  IDE_LOOP             ; "K"
            DW  SET_LBA              ; "L"  Set LBA value (Set Track,sector)
            DW  ERR                  ; "M"
            DW  SPINDOWN             ; "N"  Power down hard disk command
            DW  DRIVE_ID             ; "O"  Show current Drive ID
            DW  ERR                  ; "P"
            DW  LBA_DISPLAY_TEST     ; "Q"  Check the LBA mode HEX display on the IDE board is working correctly
            DW  READ_SEC             ; "R"  Read sector to data buffer
            DW  SEQ_SEC_RD           ; "S"  Sequential sec read and display contents
            DW  ERR                  ; "T"
```

```

DW  SPINUP          ; "U"  Power up hard disk command
DW  N_RD_SEC        ; "V"  Read N sectors
DW  WRITE_SEC       ; "W"  Write data buffer to current sector
DW  N_WR_SEC        ; "X"  Write N sectors
DW  COPY_AB         ; "Y"  Copy Drive A to Drive B
DW  VERIFY_AB       ; "Z"  Verify Drive A:= Drive B:

```

```

;IBM_BIOS COMMAND BRANCH TABLE

```

```

IBM_TABLE  dw      MENU_TIMER_TEST ;A
           dw      SET_CO_FLAG   ;B      Set Console output to Propeller or CGA/VGA Video board
           dw      MENU_FBOOT_DOS ;C      Boot MS-DOS from 5" floppy (Allow Debugging)
           dw      DEBUG_ON_OFF ;D
           dw      MENU_KEY_TEST ;E
           dw      MENU_CO_TEST  ;F
           dw      MENU_BUFF_IO ;G
           dw      XY_VIDEO      ;H
           dw      PrintScrTest  ;I      Print Screen test
           dw      READ_BYTE_TEST ;J      Tests to test/see hardware RAM RD/WR signals
           dw      READ_WORD_TEST ;K
           dw      WRITE_BYTE_TEST ;L
           dw      WRITE_WORD_TEST ;M
           dw      ERR           ;N
           dw      MENU_SIO_TEST ;O
           dw      MENU_HBOOT_DOS ;P      Boot MS-DOS from HDISK (Allow Debugging)

```

```

dw      CHS_DISPLAY_TEST ;"Q"      Check the CHS mode HEX display on the IDE board is working correctly
dw      ERR                      ;R
dw      FSEQ_5RD_TEST ;S
dw      FSEQ_3RD_TEST ;T
dw      HSEQ_RD_TEST ;U
dw      ERR                      ;V
dw      HSEC_RW_TEST ;W      Hard Disk Sector Read/Write test using INT 13H
dw      ERR                      ;X
dw      DUMP_B_SEC ;Y      Display Floppy Boot sector info
dw      DUMP_MBR ;Z      Display the Hard Disk MBR information

```

```

;Initilization table for ZILOG SCC registers (For XMODEM Input)

```

```

SCCINIT      DB      04H                      ;1, Point to WR4
              DB      44H                      ;2, X16 clock,1 Stop,NP
;
              DB      03H                      ;3, Point to WR3
              DB      0C1H                      ;4, Enable reciever, No Auto Enable (Hardware CTS), Recieve 8 bits
;              DB      0E1H                      ;4, Enable reciever, Auto Enable, Recieve 8 bits (for CTS bit)
;
              DB      05H                      ;5, Point to WR5
              DB      0EAH                      ;6, Enable, Transmit 8 bits
;              ; Set RTS,DTR, Enable
;
              DB      0BH                      ;7, Point to WR11

```

```

        DB      56H                ;8, Recieve/transmit clock = BRG
;
        DB      0CH                ;9, Point to WR12
        DB      02H                ;10, Low byte 38,400 Baud
;
        DB      06H                ;10, Low byte 19,200 Baud <<<<<<<<<
;
        DB      0EH                ;10, Low byte 9600 Baud
;
        DB      1EH                ;10, Low byte 4800 Baud
;
        DB      7EH                ;10, Low byte 1200 Baud for debugging.
;
        DB      0FEH               ;10, Low byte 300 Baud for debugging.
        DB      0DH                ;11, Point to WR13
        DB      00H                ;12, High byte for Baud
;
        DB      01H                ;12, High byte for Baud
;
        DB      0EH                ;13, Point to WR14
        DB      01H                ;14, Use 4.9152 MHz Clock. Note SD Systems uses a 2.4576 MHz clock, enable BRG
;
        DB      0FH                ;15, Point to WR15
        DB      00H                ;16, Generate Int with CTS going high

SIGNON      db      SCROLL,QUIT,BELL,CR,LF,LF,

%if      CPU_80286                ;NASM does not seem to have a "%else if"
        db      '80286'
%endif

```



```

    %if    CPU_8088
        db    '8088'
    %endif

    %if    CPU_8086
        db    '8086'
    %endif

    DB      ' Monitor V10.33 (7/25/2014) $'

MSG      DB      'THE'

    %if    CPU_80286
        DB      '80 2 86'
    %endif

    %if    CPU_8088
        DB      '80 88'
    %endif

    %if    CPU_8086
        DB      '80 86'
    %endif

    DB      ' ROM MONITOR VERSION 10.33 IS NOW ACTIVE$'

CLEANUP      DB      CR,LF,BELL,'>$'

SHOWSTACK   DB      'Stack pointer = $'

TO_BE_DONE  DB      CR,LF,'Code not done yet! ',CR,LF,'$'

```

```

AXMSG      DB      'AX=$ '
BXMSG      DB      'H  BX=$ '
CXMSG      DB      'H  CX=$ '
DXMSG      DB      'H  DX=$ '
DIMSG      DB      'H  DI=$ '
SIMSG      DB      'H  SI=$ '
CSMSG      DB      'CS=$ '
SPMSG      DB      'H  SP=$ '
SSMSG      DB      'H  SS=$ '
DSMSG      DB      'H  DS=$ '
ESMSG      DB      'H  ES=$ '
BPMSG      DB      'H  BP=$ '
H_MSG      DB      'H$'
INT_FLAGS_MSG DB    'H  Flags=$'
AddressErrMsg DB    CR,LF,'Address paramater error.$'
RangeErrMsg DB    CR,LF,'Paramater range error.$'

MAIN_MENU  DB      CR,LF
           DB      'A=Memmap    B=Video    C=DOS(F)    D=Disp RAM    E=Time & Date',CR,LF
           DB      'F=Fill RAM  G=Goto     H=Math      I=Interrupts J=Test RAM',CR,LF
           DB      'K=Menu      L=8259A    M=Move RAM   N=IDE Menu   O=',CR,LF
           DB      'P=DOS(H)    Q=Ports   R=Registers S=Subs      T=Disp RAM (Words)',CR,LF
           DB      'U=All Ports V=Verify   W=XModem    X=PC-BIOS   Z=Z80',CR,LF,'$'

DIFF_Header_Msg DB  CR,LF,'First RAM    HEX Binary      Second RAM    HEX Binary$'

```

```

MATCHES_OK      DB      CR,LF,'Both RAM locations match$'

PORTS_IN_MSG    DB      CR,LF,'Input Ports, 0-0FFFFH Ports.  (16 Bit Port DX->AX and DX->AL) ',CR,LF,LF,'$'

MORE_MSG        DB      CR,LF,'Continue ? (Y/N) $'

MSG30           DB      CR,LF,'Adj :- $'

MSG12T          DB      '      $'

MSG16T          DB      '/20$'

JMSG            DB      CR,LF,LF,'Continous RAM hardware test.',CR,LF,'Please enter start , ending address (+CR).',CR,LF,'$'

STARTJMSG       DB      CR,LF,'Starting RAM test. Hit ESC any time to abort',CR,LF,'$'

RAM_Test_Count  DB      CR,'RAM test loop count = $'

TMSG            DB      CR,LF,'Time:- $'

GET_SEG_MSG     DB      CR,LF,'Enter Segment (xxxxH)->$'

GET_OFFS_MSG    DB      CR,LF,'Enter Offset (xxxxH)->$'

MATH_MSG        DB      CR,LF,LF,'Hex Math. Enter xxxxH,xxxxH:- $'

MATH_HEADER1    DB      CR,LF,'Sum = $'

MATH_HEADER2    DB      'H.      Difference = $'

PIC_SIGNON      DB      CR,LF,LF,'Test of Interrupts on the MSDOS Support Board',CR,LF
                DB      'Any keyboard key should flash the "D1" LED.',CR,LF
                DB      'If CPU returned sINTA, "D2" LED should flash.',CR,LF,LF,'$'

CRLFMSG         DB      CR,LF,'$'

TrapIntMSG      DB      'Trap int. detected at a non-assigned location.$'

TrapFFIntMSG    DB      'Trap int. detected at 0FFH in RAM.$'

DebugTrapMSG    DB      'Trap int. detected Software Debug INT at 0CH in RAM.$'

Int0MSG         DB      'V0 $'

Int1MSG         DB      'V1 $'

Int2MSG         DB      'V2 $'

```

```

Int3MSG      DB      'V3 $'

Int4MSG      DB      'V4 $'

Int5MSG      DB      'V5 $'

Int6MSG      DB      'V6 $'

Int7MSG      DB      'V7 $'


DIVIDE_MSG    DB      CR,LF,'Int #0 (Invalid Divide), FAULT',CR,LF,'$'

PM_DIVIDE_MSG DB      'PM Int 0 Divide, FAULT',CR,LF,'$'

DEBUG_MSG     DB      'Int 1 Debug Exception, TRAP',CR,LF,LF,'$'

BREAKPOINT_MSG DB      'Int 3 Breakpoint, TRAP',CR,LF,LF,'$'

NMI_FAULT_MSG DB      CR,LF,'Int #2 (NMI Interrupt), TRAP',CR,LF,'$'

OVERFLOW_ERR_MSG DB    CR,LF,'Int #4 (Overflow), TRAP',CR,LF,'$'

BOUNDS_ERR_MSG DB      CR,LF,'Int #5 (Bounds Check), FAULT',CR,LF,'$'

INVALID_ERR_MSG DB      CR,LF,'Int #6 (Invalid Opcode), FAULT',CR,LF,'$'

DEVICE_ERR_MSG DB      CR,LF,'Int #7 (No Coprocessor), FAULT',CR,LF,'$'

DOUBLE_ERR_MSG DB      CR,LF,'Int #8 (Double Fault), ABORT',CR,LF,'$'

COPROCESSOR_SEG_ERR_MSG DB CR,LF,'Int #9 Co-processor Segment Overrun, ABORT',CR,LF,'$'

INVALID_TSS_ERR_MSG DB    CR,LF,'Int #10 Invalid TSS, FAULT,  $'

INT_ERR_NUM_MSG DB      'Error Number = $'

SEGMENT_ERR_MSG DB      CR,LF,'Int #11 Segment not present, FAULT,  $'

STACK_ERR_MSG DB      CR,LF,'Int #12, Stack exception, FAULT $'

GENERAL_ERR_MSG DB      CR,LF,'Int #13 General protection error,  FAULT  $'

PAGE_ERR_MSG  DB      CR,LF,'Int #14 Page Fault,  FAULT',CR,LF,'$'

RESERVED_ERR_MSG DB    CR,LF,'INT #15 Intel Reserved Int, TRAP',CR,LF,'$'

COPROCESSOR_ERR_MSG DB    CR,LF,'Int #16 Co-processor error,  FAULT',CR,LF,'$'

```

```

CPU_HALTED_MSG      DB      'The CPU is Halted. Press Reset to restart.',CR,LF,'$'

INT_SIGNON          DB      CR,LF,LF,'Test of CPU FAULT & TRAP Interrupts.'
                    DB      CR,LF,'Note. The CPU will HALT upon accepting FAULTS!'
                    DB      CR,LF,LF,'CPU Interrupt Testing Menu',CR,LF
                    DB      '00 = Divide by Zero FAULT test',CR,LF
                    DB      '04 = Overflow TRAP test',CR,LF
                    DB      '06 = Invalid Opcode FAULT test',CR,LF
                    DB      '0D = General Protection Fault, FAULT test',CR,LF
                    DB      '40 = Software Int 40H test',CR,LF
                    DB      'F0 = Software Int F0H test',CR,LF,LF

INT_RANGE_MSG DB      CR,LF,BELL,'Only CPU ints 00, 04, 06, 0D, 40 or F0 for this test.',CR,LF,'$'

MODEM_SIGNON: DB      'Load a File from a PC into RAM using the S100Computers IO Board',CR,LF
                    DB      'Zilog SCC Ports A1H & A3H. Requires RTS & CTS, 38,400 Baud.',CR,LF,'$'

SSC_MSG_INIT DB      'SCC Port A initilized to 38,400 Baud.',CR,LF,LF,'$'

RAM_DESTINATION      DB      CR,LF,'Enter destination in RAM for data (up to 5 digits): $'

P_RAM_DESTINATION DB      CR,LF,'Enter destination in RAM for data (up to 8 digits): $'

DOWNLOAD_MSG DB      'Downloading file Started.$'

SPEED_MSG            DB      'Serial Port is set to 38,400 Baaud$'

RMSG                 DB      CR,LF,'WAITING FOR SECTOR # $'

ERRSOH               DB      'H Received',CR,LF,'Did not get Correct SOH',CR,LF,'$'

MODEM_ERR2           DB      CR,LF,'Bad Sector # in Header',CR,LF,'$'

MODEM_ERR3           DB      CR,LF,'Bad Checksum for Sector',CR,LF,'$'

```

```

TOUTM      DB      CR,LF,'Timeout! $'

QUITM      DB      CR,LF,'+++ MULTIPLE ERRORS ENCOUNTERED +++'

           DB      CR,LF,'Type Q To Quit, R To Retry:$'

RAM_MSG    DB      'H.  If OK will write to RAM location $'

FINISH_MSG DB      'Down loading of file complete.  No Errors$'

TRANS_DONE DB      CR,LF,LF,'Data Transfer Is Complete',CR,LF,LF,'$'

BAD_HEADER DB      CR,LF,'Unable to get a valid file header!',CR,LF,'$'

ENTER_RAM_LOC DB    CR,LF,'Enter start RAM destination (1000H - FF000H) (xx000H): $'

START_POINTER DB    CR,LF,'Will load data starting at RAM location $'

ABORT_MSG  DB      CR,LF,LF,'Invalid Character or Program Aborted',CR,LF,'$'


IDE_SIGNON0 DB      CR,LF,LF,'IDE HDisk Test Menu Routines.  $'

IDE_SIGNON4 DB      'A=Select Drive A  B=Select Drive B  C=Boot CPM  D=Set Sec Display $'

IDE_SIGNON1 DB      'On',CR,LF,'$'

IDE_SIGNON2 DB      'Off',CR,LF,'$'


IDE_SIGNON3 DB      'E=Clear Sec Buff  F=Format Disk      I=Next Sec  J=Previous Sec',CR,LF

           DB      'L=Set LBA Value  N=Power Down      O=Disk ID  Q=LBA Display Test',CR,LF

           DB      'R=Read Sector    S=Seq Sec Rd      U=Power Up  V=Read N Sectors',CR,LF

           DB      'W=Write Sector    X=Write N Sectors Y=Copy A->B  Z=Verify A=B',CR,LF

           DB      '(ESC) Back to Main Menu',CR,LF

           DB      LF,'Current settings:- $'


IDE_MENU   DB      'Enter a Command:- $'

```

```

IDE_HARDWARE DB      CR,LF,'Initilizing IDE Board, one moment please...',CR,LF,'$'

INIT_1_ERROR: DB     CR,LF,'Initilizing of First Drive failed. Aborting Command.',BELL,CR,LF,LF,'$'
INIT_2_ERROR DB     CR,LF,'Initilizing of Second Drive failed. (Possibly not present).',BELL,CR,LF,LF,'$'
BAD_DRIVE:   DB     CR,LF,'First Drive ID Information appears invalid. '
              DB     '(Drive possibly not present).',CR,LF
              DB     'Aborting Command.',BELL,CR,LF,LF,'$'

msgmdl       DB     CR,LF,'Drive/CF Card Information:-',CR,LF
              DB     'Model: $'
msgsn        DB     'S/N:  $'
msgrev       DB     'Rev:   $'
msgcy        DB     'Cylinders: $'
msghd        DB     ', Heads: $'
msgsc        DB     ', Sectors: $'
msgCPMTRK    DB     'CPM TRK = $'
msgCPMSEC    DB     ' CPM SEC = $'
msgLBA       DB     ' (LBA = 00$'
MSGBracket   DB     ')$'
H_Msg        DB     'H$'
H_MSG_CRLF   DB     'H',CR,LF,'$'
NotDoneYet   DB     CR,LF,'Command Not Done Yet$'
CONFIRM_WR_MSG DB     CR,LF,LF,BELL,'Will erase data on the current drive, '
              DB     'are you sure? (Y/N)...$'
msgrd        DB     'Sector Read OK',CR,LF,'$'

```

```

msgwr      DB      'Sector Write OK',CR,LF,'$'

SET_LBA_MSG DB      'Enter CPM style TRK & SEC values (in hex).',CR,LF,'$'

SEC_RW_ERROR DB      'Drive Error, Status Register = $'

ERR_REG_DATA DB      'Drive Error, Error Register = $'

ENTERRAM_SECL DB      'Starting sector number,(xxH) = $'

ENTERRAM_HEAD DB      'Starting HEAD number,(xxH) = $'

ENTERRAM_FTRKL DB      'Enter Starting Track number,(xxH) = $'

ENTERRAM_TRKL DB      'Track number (LOW byte, xxH) = $'

ENTERRAM_TRKH DB      'Track number (HIGH byte, xxH) = $'

ENTER_HEAD  DB      'Head number (01-0f) = $'

ENTER_COUNT DB      'Number of sectors to R/W (xxH) = $'

ENTERRAM_DMA DB      'Enter DMA Adress (Up to 5 digits, xxxxxH) = $'

OVER_COUNT_10 DB      CR,LF,'1 & 9 sectors. Only!',CR,LF,'$'

OVER_COUNT_19 DB      CR,LF,'1 & 18 sectors. Only!',CR,LF,'$'

DRIVE_BUSY   DB      'Drive Busy (bit 7) stuck high.   Status = $'

DRIVE_NOT_READY DB      'Drive Ready (bit 6) stuck low.   Status = $'

DRIVE_WR_FAULT DB      'Drive write fault.   Status = $'

UNKNOWN_ERROR DB      'Unknown error in status register.   Status = $'

BAD_BLOCK    DB      'Bad Sector ID.   Error Register = $'

UNRECOVER_ERR DB      'Uncorrectable data error.   Error Register = $'

READ_ID_ERROR DB      'Error setting up to read Drive ID',CR,LF,'$'

SEC_NOT_FOUND DB      'Sector not found. Error Register = $'

INVALID_CMD  DB      'Invalid Command. Error Register = $'

TRK0_ERR     DB      'Track Zero not found. Error Register = $'

UNKNOWN_ERROR1 DB      'Unknown Error. Error Register = $'

```



```

CONTINUE_MSG DB    CR,LF,'To Abort enter ESC. Any other key to continue. $'

FORMAT_MSG_A DB    'Fill disk sectors of Disk [A] with 0E5H$'

FORMAT_MSG_B DB    'Fill disk sectors of Disk [B] with 0E5H$'

ATHOME_MSG   DB    CR,LF,BELL,'Already on Track 0, Sector 0$'

AT_START_MSG DB    CR,LF,BELL,'Already at start of disk!$'

AT_END_MSG   DB    CR,LF,BELL,'At end of Disk!$'

FILL_MSG     DB    CR,LF,'Sector buffer area cleared to 0000....$'

READN_MSG    DB    CR,LF,'Read multiple sectors from current disk/CF card to RAM buffer.'
              DB    CR,LF,'How many 512 byte sectores (xx HEX):$'

WRITEN_MSG   DB    CR,LF,'Write multiple sectors RAM buffer CURRENT disk/CF card.'
              DB    CR,LF,'How many 512 byte sectores (xx HEX):$'

READN_S_MSG  DB    CR,LF,'Read Sector to RAM buffer. $'

WRITEN_S_MSG DB    CR,LF,'Write Sector from RAM buffer. $'


DiskCopyMsg  DB    CR,LF,'Copy CPM Partition on Drive A to Drive B (Y/N)? $'

DiskVerifyMsgDB    CR,LF,'Will verify CPM Partition on Drive A to Drive B.$'

CopyDone     DB    CR,LF,'Disk Copy Done.$'

VERIFY_ERR   DB    CR,LF,BELL,'Verify Error. $'

VerifyDone   DB    CR,LF,'Disk Verify Done.$'

CR_To_Continue DB    CR,LF,'Hit any key to continue.$'

OK_CR_MSG    DB    ' OK',CR,LF,'$'

COPY_ERR     DB    CR,LF,BELL,'Sector Copy Error.$'

CURRENT_MSG_A DB    ' <<<<< Current Drive = [A] >>>>>',CR,LF,LF,'$'

CURRENT_MSG_B DB    ' <<<<< Current Drive = [B] >>>>>',CR,LF,LF,'$'

FORMAT_ERR   DB    CR,LF,BELL,'Sector Format Error$'

```

```

ERR_MSG          DB      CR,LF,BELL,'Invalid Command (or code not yet done)',CR,LF,'$'

IBM_SIGNON_MSG   DB      CR,LF,LF,'IBM PC BIOS Initilizing$'

IBM_MENU1        DB      CR,LF,LF,'IBM-PC BIOS Test Menu.      (Debug Flag = $'
IBM_MENU_ON      DB      'ON)',CR,LF,'$'
IBM_MENU_OFF     DB      'OFF)',CR,LF,'$'

IBM_MENU2        DB      'A=Timer Test          B=Propeller/LAVA/VGA    C=MS-DOS Boot (Floppy)',CR,LF
                  DB      'D=Toggle Debug Flag    E=Key Press Test      F=Consol Out Test',CR,LF
                  DB      'G=Keyboard Buffer Test   H=INT 10H CMDs          I=Print Screen Test',CR,LF
                  DB      'J=RAM Byte READ Test    K=RAM Word READ Test    L=RAM Byte WRITE Test',CR,LF
                  DB      'M=RAM Word WRITE Test   O=Out to Serial Port    P=MS-DOS Boot (HDISK)',CR,LF
                  DB      'Q=CHS Hex Display Test   S=5" Floppy Sec RD Test T=3" Floppy Sec RD Test',CR,LF
                  DB      'U=HDisk Sec RD Test      W=HDisk Sector R/W Test Y=Floppy Boot Sec Info',CR,LF
                  DB      'Z=Hard Disk MBR Info      (ESC) Back to Main Menu',CR,LF,LF,'>$'

NMI_MSG          DB      CR,LF,BELL,'Recieved an NMI Interrupt.',CR,LF,'$'

ZFDC_FAIL_MSG    DB      CR,LF,BELL,'ZFDC Board failed to initilize',CR,LF,'$'
ZFDC_OK_MSG      DB      CR,LF,'ZFDC Board Initilize OK',CR,LF,'$'
PIC_INIT_MSG     DB      CR,LF,'Initilizing 8259A PIC (Port 20H, Ints 0 & 1 only)$'
RESET_FAIL_MSG   DB      CR,LF,BELL,'Reset of floppy drive failed.',CR,LF,'$'
HRESET_FAIL_MSG  DB      CR,LF,BELL,'Reset of Hard Disk drive failed.',CR,LF,'$'
BOOT_FAIL_MSG    DB      CR,LF,BELL,'Boot sector read on floppy drive failed.',CR,LF,'$'
BOOT_OK_MSG      DB      CR,LF,'Boot Sector Loader Signature Valid (AA55H).',CR,LF,'Now Booting  MS-DOS.....',CR,LF,LF,'$'
READ_ERR_MSG     DB      CR,LF,BELL,'Floppy Sector Read Error. Error returned = $'

```

```

WRITE_ERR_MSG DB      CR,LF,BELL,'Floppy Sector Write Error. Error returned = $'

HREAD_ERR_MSG DB      CR,LF,BELL,'HDisk Multi-Sector Read Error.$'

HWRITE_ERR_MSG  DB      CR,LF,BELL,'HDisk Multi-Sector Write Error.$'

HOME_ERR_MSG  DB      CR,LF,BELL,'Disk reset error.',CR,LF,'$'

NO_BASIC_MSG  DB      CR,LF,BELL,'BASIC Handler error.',CR,LF,'$'

NO_BREAK_MSG  DB      CR,LF,BELL,'Keyboard Break Handler error.',CR,LF,'$'

NO_COMM_MSG   DB      CR,LF,BELL,'Serial Communications Handler error.',CR,LF,'$'

CASSETTE_MSG  DB      CR,LF,BELL,'Cassette Handler error. AH=$'

FBOOT_DOS_MSG DB      CR,LF,'Booting MS-DOS from 5" Floppy Disk$'

HBOOT_DOS_MSG DB      CR,LF,'Booting MS-DOS from HARD Disk$'

KEY_TEST_MSG  DB      CR,LF,'Software Interrupt driven Keyboard Input test (ESC to Abort)',CR,LF,'$'

IN_CHAR_MSG   DB      CR,LF,'Type one character',CR,LF,'$'

GOT_CHAR_MSG  DB      'H <--- Hex value of character recieved via software Int 16H.',CR,LF,'$'

CO_TEST_MSG   DB      CR,LF,'Software Interrupt driven Console/Video out test',CR,LF,'$'

OUT_CHAR_MSG  DB      '<--- Character Recieved$'

TIMER_TEST_MSG DB      CR,LF,'8259A Interrupt driven Timer Test$'

TIMER_DATA_MSG DB      CR,LF,'Enter any key to read timer data. (ESC to Abort)$'

TIMER_LOW_MSG DB      CR,LF,'Timer Low Value = $'

TIMER_HIGH_MSG DB      'H',CR,LF,'Timer High Value = $'

TIMER_OFLOW_MSG DB      'H',CR,LF,'Timer Overflow Value = $'

BUFF_TEST_MSG DB      CR,LF,'Type keyboard characters as fast as you can!',CR,LF,'$'

SQRDHFAILMSG  DB      CR,LF,BELL,'Error reading sectors from HARD disk',CR,LF,'$'

SQRD5FAILMSG  DB      CR,LF,BELL,'Error reading sectors from 5" Floppy',CR,LF,'$'

SQRD3FAILMSG  DB      CR,LF,BELL,'Error reading sectors from 3" Floppy',CR,LF,'$'

SQRDHOKMSG    DB      CR,LF,'Read sectors from HARD disk OK!',CR,LF,'$'

```

```

SQRD50KMSG    DB      CR,LF,'Read sectors from 5" 360K Floppy disk OK!$'

SQRD30KMSG    DB      CR,LF,'Read sectors from 3" 1.44M Floppy disk OK! ',CR,LF,'$'

DEBUG_SET_MSG DB      CR,LF,'Set Debug level (0 = OFF, 1 = INTs only, 2 = +HDisk Info, 3 = +Floppy Info) $'

DUMP_ON1_MSG  DB      CR,LF,'Debug flag ON (Level 1)',CR,LF,'$'

DUMP_ON2_MSG  DB      CR,LF,'Debug flag ON (Level 2)',CR,LF,'$'

DUMP_ON3_MSG  DB      CR,LF,'Debug flag ON (Level 3)',CR,LF,'$'

DUMP_OFF_MSG  DB      CR,LF,'Debug flag OFF',CR,LF,'$'

SEC_5RD_MSG   DB      CR,LF,'Sequentially read sectors from 5" Floppy disk',CR,LF,'$'

SEC_3RD_MSG   DB      CR,LF,'Sequentially read sectors from 3" Floppy disk',CR,LF,'$'

SEC_HDRD_MSG  DB      CR,LF,'Read sector from HARD Disk test using Int 13H',CR,LF,'$'

NOT_DONE_MSG  DB      CR,LF,BELL,'Code Not done yet',CR,LF,'$'

INVALID_AH_FMSG DB      CR,LF,BELL,'Invalid AH paramater in Floppy Handler. AH=$'

INVALID_AH_HMSG DB      CR,LF,BELL,'Invalid AH paramater in HDisk Handler. AH=$'

SIO_TEST_MSG  DB      CR,LF,'Serial Port (A3H) Test.'

                DB      CR,LF,'Enter any text. (ESC to stop). ',CR,LF,'>','$'

SIO_INIT_ERR  DB      CR,LF,'Serial Port Initilization Error. AH=$'

SIO_ERR       DB      CR,LF,'Error sending character to Serial Port. AH=$'

INT_13F_MSG   DB      CR,LF,'Int 13H (Floppy)$'

INT_40F_MSG   DB      CR,LF,'Int 40H (<--Floppy)$'

INT_13H_MSG   DB      CR,LF,'Int 13H (*HDisk*)$'


INT_AX_MSG    DB      CR,LF,'AX=$'

INT_BX_MSG    DB      'H  BX=$'

INT_CX_MSG    DB      'H  CX=$'

INT_DX_MSG    DB      'H  DX=$'

```

```

INT_SI_MSG      DB      'H',CR,LF,'SI=$'

INT_DI_MSG      DB      'H  DI=$'

INT_BP_MSG      DB      'H  BP=$'

INT_SP_MSG      DB      'H  SP=$'

INT_CS_MSG      DB      CR,LF,'CS=$'

INT_DS_MSG      DB      'H  DS=$'

INT_ES_MSG      DB      'H  ES=$'

INT_SS_MSG      DB      'H  SS=$'

INT_FS_MSG      DB      'H  FS=$'

INT_GS_MSG      DB      'H  GS=$'

IP_ADDRESS_MSG  DB      'IP=$'


INT_1AH_MSG     DB      CR,LF,'Int 1AH (Time)$'

INT_10H_MSG     DB      CR,LF,'Int 10H (VIDEO)$'

INT_15H_MSG     DB      CR,LF,'Int 15H (Cassette)$'

SIDE_REQUEST_MSG DB CR,LF,'Read from Side A or Side B (A/B) $'

SIDE_A_SET_MSG  DB      CR,LF,'Will read from Side A',CR,LF,'$'

SIDE_B_SET_MSG  DB      CR,LF,'Will read from Side B',CR,LF,'$'

FORMAT_ERR_MSG  DB      CR,LF,'ZFDC Track Format error $'


CMOS_CLOCK_MSG  DB      CR,LF,BELL,'CMOS RTC Error',CR,LF,'$'

CMOS_STUCK_MSG  DB      CR,LF,BELL,'CMOS RTC "Stuck" Error',CR,LF,'$'

CMOS_RANGE_MSG  DB      CR,LF,BELL,'CMOS RTC Incorrect BCD values Error',CR,LF,'$'

SECTOR_NUM_MSG  DB      CR,LF,'Starting requested Sector = $'

HRESET_OK_MSG DB CR,LF,'Reset of Hard Disk drive OK.',CR,LF,'$'

```

```

RD_ERR_MSG    DB      CR,LF,BELL,'Sector READ Error Returned.'

               DB      CR,LF,'Head = $'

TRACK_MSG     DB      'H   Track = $'

SEC_MSG       DB      'H   Sector = $'

WR_ERR_MSG    DB      CR,LF,BELL,'Sector WRITE Error Returned.'

               DB      CR,LF,'Head = $'

ESC_END_MSG   DB      CR,LF,'Press ESC to Abort. Any other key to continue $'

SEQAT500      DB      CR,LF,LF,'Sector(s) loaded @ 0000:500H.'

               DB      CR,LF,'Head = $'

CR_TAB_MSG    DB      CR,LF,'                               $'

LBA_TEST_MSG  DB      CR,LF,'Test for LBA on IDE drive #2 (using LBA mode)$'

CHS_TEST_MSG  DB      CR,LF,'Test for CHS on IDE drive #2 (using non-LBA mode)$'


TRKL_NUM      DB      CR,LF,'Enter TRACK/Cylinder number (LOW byte, xxH) = $'

TRKH_NUM      DB      CR,LF,'Enter TRACK/Cylinder number (HIGH byte, xxH) = $'

HEAD_NUM      DB      CR,LF,'Enter HEAD number, (0-FH, 0xH) = $'

SECTOR_NUM    DB      CR,LF,'Enter SECTOR number (xxH) = $'

CHECK_DISPLAY_MSG DB      CR,LF,'Check the IDE Board HEX display.$'

BOOT_3RD_MSG  DB      CR,LF,'Display Floppy Boot Sector Information.',CR,LF,'$'

DRIVE_SELECT_MSG DB CR,LF,'Please select floppy disk (A or B) $'

BOOT_INFO_FAIL_MSG DB      CR,LF,'Error reading Boot disk sector.$',CR,LF

BOOT_INFOOKMSG    DB      CR,LF,'Floppy Boot Sector Information:-',CR,LF,LF,'$'


JMP_MSG       DB      '      Boot JMP Vector',CR,LF,'$'

NAME_MSG      DB      '      OEM Name',CR,LF,'$'

```

```

BYTES_MSG      DB      '      Bytes/Sec',CR,LF,'$'
CLUSTER_MSG    DB      '      Sec/Cluster',CR,LF,'$'
RES_MSG        DB      '      Reserved Sectors',CR,LF,'$'
FATS_MSG       DB      '      FATS',CR,LF,'$'
ROOT_MSG       DB      '      Root Dir Entries',CR,LF,'$'
SECTORS_MSG    DB      '      Sectors',CR,LF,'$'
MEDIA_MSG      DB      '      Media Byte',CR,LF,'$'
FAT_SEC_MSG    DB      '      FAT Sectors',CR,LF,'$'
SEC_TRK_MSG    DB      '      Sectors/Track',CR,LF,'$'
HEADS_MSG      DB      '      Heads',CR,LF,'$'
HIDDEN_MSG     DB      '      Hidden Sectors',CR,LF,'$'
HUGE_MSG       DB      '      Huge Sectors',CR,LF,'$'
DRIVE_NO_MSG   DB      '      Drive #',CR,LF,'$'
RESERVED_MSG   DB      '      Reserved',CR,LF,'$'
BOOT_SIG_MSG   DB      '      Boot Signature',CR,LF,'$'
VOL_ID_MSG     DB      '      Volume ID',CR,LF,'$'
VOLUME_MSG     DB      '      Volume Label',CR,LF,'$'
SYS_TYPE_MSG   DB      '      File Sys Type',CR,LF,LF,'$'
NO_MBL_MSG     DB      CR,LF,BELL,'Invalid Floppy Boot Loader Signature detected',CR,LF,'$'
BOOT_MBR_MSG   DB      CR,LF,'Reading Hard Disk MBR sector, (C=0, H=0, S=1)',CR,LF,'$'
BOOT_MBR_FAIL_MSG DB    CR,LF,BELL,'Error reading Hard Disk MBR sector',CR,LF,'0'
MBR_INFOOKMSG  DB      'Hard Disk Master Boot Record:-',CR,LF,'$'

DISK_SIG_MSG   DB      '      Hard Disk Signature',CR,LF,'$'
NULS_MSG       DB      '      Usually Nulls (Optional)',CR,LF,LF,'$'

```

```

PT1_MSG          DB      ' First Partition Table',CR,LF,'$'
PT2_MSG          DB      ' Second Partition Table',CR,LF,'$'
PT3_MSG          DB      ' Third Partition Table',CR,LF,'$'
PT4_MSG          DB      ' Forth Partition Table',CR,LF,'$'
SIGNATURE_MSG DB      '      LBR Signature Word',CR,LF,'$'
STATUS_MSG      DB      ' Status Byte, $'
STLBA_MSG       DB      ' Start CHS Address, $'
PAR_TYPE_MSG    DB      ' Partition Type, $'
ECHS_MSG        DB      ' End CHS Address',CR,LF,'$'
SLB_MSG         DB      ' Start LBA Address, $'
ELBA_MSG        DB      ' End LBA Address',CR,LF,LF,'$'
CYL_MSG         DB      'H Cyl=$'
HD_MSG          DB      ' Head=$'
BRAC1_MSG       DB      'H ($'
OF_MSG          DB      'H of $'
BRAC2_MSG       DB      'H)',CR,LF,'$'
DRIVE1_MSG      DB      ' on Drive A',CR,LF,'$'
DRIVE2_MSG      DB      ' on Drive B',CR,LF,'$'
HRW_TEST_MSG    DB      CR,LF,'Hard Disk Sector R/W test using INT 13H',CR,LF
                DB      CR,LF,'>>> WARNING <<< Data on Disk will be overwritten. Continue...(Y/N) $'
ONE_MOMENT_MSG  DB      CR,LF,'One moment while IDE Drive is being initilized',CR,LF,'$'
ASK_WR_MSG      DB      CR,LF,'Write data back to Hard Disk...(Y/N)$'
START_DATA_MSG  DB      'H Start of Data =',CR,LF,'$'
SEC_READ_OK     DB      CR,LF,'Sector(s) read OK',CR,LF,'$'
SEC_BACK_OK     DB      CR,LF,'Sector(s) written back OK',CR,LF,'$'

```



```

LOOP_ESC_MSG DB      CR,LF,'Will R/W sectors until ESC is entered. CR to start',CR,LF,'$'

VIDIO_TEST_MSG      DB      CR,LF,'Int 10H tests for control of Video Board I/O',CR,LF
                    DB      CR,LF,'Enter value of [AX], (xxxxH) $'
ENTER_BX_MSG DB      CR,LF,'Enter value of [BX], (xxxxH) $'
ENTER_CX_MSG DB      CR,LF,'Enter value of [CX], (xxxxH) $'
ENTER_DX_MSG DB      CR,LF,'Enter value of [DX], (xxxxH) $'
ACTIVATE_INT_MSG DB  CR,LF,'Enter CR to implement the INT 10H interrupt, (ESC to Abort) $'

VID_PARM_TBD_MSG DB  CR,LF,'Int 10H Video paramater not yet implemented'
                    DB      CR,LF,'                ','$'
VID_PARM_TBD1_MSG DB      CR,LF,'Int 10H Video paramater not fully completed'
                    DB      CR,LF,'                ','$'
VID_PARM_MSG DB      CR,LF,'Invalid Int 10H Video paramater',CR,LF,'$'
VIDIO_OUTPUT_MSG DB  CR,LF,'    Set MSDOS (INT 10H) Console Output Menu'
                    DB      CR,LF,'    0 = MSDOS output will be to Propeller Board.'
                    DB      CR,LF,'    1 = MSDOS Output will be to CGA/VGA Board.'
                    DB      CR,LF,'    2 = MSDOS Output will be to LAVA Board.'
                    DB      CR,LF,'Please enter selection. (ESC to abort):$'
VIDIO_LAVA_MSG      DB      CR,LF,'MSDOS output will be to LAVA Board',CR,LF,'$'
VIDIO_PROP_SMSG      DB      CR,LF,'MSDOS output will be to Propeller Board',CR,LF,'$'
VIDIO_VGA_MSG DB      CR,LF,'MSDOS output will be to CGA/VGA Board',CR,LF,'$'
VIDIO_PROP_MSG      DB      'MSDOS output will be to Propeller Board$'
VIDIO_VGA_SMSG      DB      'MSDOS output will be to CGA/VGA Board$'
VIDIO_LAVA_SMSG      DB      'MSDOS output will be to LAVA Board$'

```

```

INT10_ERR_MSG DB    CR,LF,'Sorry invalid selection. Must be 0,1, or 2 $'

PSCR_TEST_MSG DB    CR,LF,'Will Print (CGA/VGA) screen on Printer',CR,LF,'$'


PSCR_TEST_LEN EQU    $-PSCR_TEST_MSG-3

PRN_INIT_STR  DB    0H                                ;<--- I cannot get the PCL-6 Codes below to work with my printer
;              DB    1BH,'E'                            ;PCL-6 reset
;              DB    1BH,'%12345X',0                    ;Universal Exit
;              DB    1BH,'&','l',01H,'O',0              ;Landscape Orientation (Cannot seem to get these to work)


TIME_ERROR_MSG DB    CR,LF,'RTC Error',CR,LF,0

Time_Msg      DB    'Time=',0

GAP_Msg       DB    ' ',0

Date_Msg      DB    'Date=',0

SET_TIME_MSG  DB    'Do you wish to set the time and date (Y/N): $'

Input_Hours_Msg      DB    CR,LF,'Please Enter Hours (2 digits, 00-24) ',0
Input_Minutes_Msg    DB    CR,LF,'Please Enter Minutes (2 digits, 00-60) ',0
Input_Seconds_Msg    DB    CR,LF,'Please Enter Seconds (2 digits, 00-60) ',0
Input_Year_Msg       DB    CR,LF,'Please Enter Year (2 digits, 20xx) 20',0
Input_Month_Msg      DB    CR,LF,'Please Enter Month (2 digits, 00-12) ',0
Input_Day_Msg        DB    CR,LF,'Please Enter day (2 digits, 01-31) ',0


ROMCHECK_MSG  DB    CR,LF,'Initilizing VGA ROM at C000:0000H',0

ROMCHECK_MSG_OK      DB    CR,LF,'VGA ROM Initilized, returned back to BIOS.',CR,LF,0

NO_VGA_MSG      DB    CR,LF,'No VGA ROM at C000:0000H',0

```

```

UNASSIGNED_1_INT_MSG      DB      CR,LF,'Un-assigned Vector Interrupt # $'

VGA_OK_MSG                DB      CR,LF,'Int 15H VGA Initilization Done',CR,LF,0

NO_8259A_MSG              DB      CR,LF,BELL,'8259A PIC Not found. (MS-DOS will not be bootable)',BELL,CR,LF,0

BYTE_RTEST_MSG            DB      CR,LF,'Location for RAM byte read test: ',0

WORD_RTEST_MSG            DB      CR,LF,'Location for RAM word read test: ',0

BYTE_WTEST_MSG            DB      CR,LF,'Location for RAM byte write test: ',0

WORD_WTEST_MSG            DB      CR,LF,'Location for RAM word write test: ',0

```

```

;*****

;   CHARACTER GENERATOR GRAPHICS FOR 320X200 AND 640X200 GRAPHICS

;*****

```

CRT_CHAR_GEN:

```

%if    CPU_80286                                ;<<<< NOTE I HAVE REMOVED THIS CODE FOR 8088/8086 Boards,

                                                ;   NOT ENOUGH WITH 28C256 EEPROMS. Not normally used anyway.

%if    MONITOR_ROM

TIMES 0FAE6H-($-$$) DB 0                        ;Locate exactly where IBM has it

%endif

```

```

DB      000H,000H,000H,000H,000H,000H,000H,000H      ;

DB      07EH,081H,0A5H,081H,0BDH,099H,081H,07EH      ;

DB      07EH,0FFH,0DBH,0FFH,0C3H,0E7H,0FFH,07EH      ;

DB      06CH,0FEH,0FEH,0FEH,07CH,038H,010H,000H      ;

DB      010H,038H,07CH,0FEH,07CH,038H,010H,000H      ;

```

```

DB      038H,07CH,038H,0FEH,0FEH,07CH,038H,07CH      ;
DB      010H,010H,038H,07CH,0FEH,07CH,038H,07CH      ;
DB      000H,000H,018H,03CH,03CH,018H,000H,000H      ;
DB      0FFH,0FFH,0E7H,0C3H,0C3H,0E7H,0FFH,0FFH      ;
DB      000H,03CH,066H,042H,042H,066H,03CH,000H      ;
DB      0FFH,0C3H,099H,0BDH,0BDH,099H,0C3H,0FFH      ;
DB      00FH,007H,00FH,07DH,0CCH,0CCH,0CCH,078H      ;
DB      03CH,066H,066H,066H,03CH,018H,07EH,018H      ;
DB      03FH,033H,03FH,030H,030H,070H,0F0H,0E0H      ;
DB      07FH,063H,07FH,063H,063H,067H,0E6H,0C0H      ;
DB      099H,05AH,03CH,0E7H,0E7H,03CH,05AH,099H      ;
DB      080H,0E0H,0F8H,0FEH,0F8H,0E0H,080H,000H      ;
DB      002H,00EH,03EH,0FEH,03EH,00EH,002H,000H      ;
DB      018H,03CH,07EH,018H,018H,07EH,03CH,018H      ;
DB      066H,066H,066H,066H,066H,000H,066H,000H      ;
DB      07FH,0DBH,0DBH,07BH,01BH,01BH,01BH,000H      ;
DB      03EH,063H,038H,06CH,06CH,038H,0CCH,078H      ;
DB      000H,000H,000H,000H,07EH,07EH,07EH,000H      ;
DB      018H,03CH,07EH,018H,07EH,03CH,018H,0FFH      ;
DB      018H,03CH,07EH,018H,018H,018H,018H,000H      ;
DB      018H,018H,018H,018H,07EH,03CH,018H,000H      ;
DB      000H,018H,00CH,0FEH,00CH,018H,000H,000H      ;
DB      000H,030H,060H,0FEH,060H,030H,000H,000H      ;
DB      000H,000H,0C0H,0C0H,0C0H,0FEH,000H,000H      ;
DB      000H,024H,066H,0FFH,066H,024H,000H,000H      ;

```

DB	000H, 018H, 03CH, 07EH, 0FFH, 0FFH, 000H, 000H	;
DB	000H, 0FFH, 0FFH, 07EH, 03CH, 018H, 000H, 000H	;
DB	000H, 000H, 000H, 000H, 000H, 000H, 000H, 000H	;
DB	030H, 078H, 078H, 030H, 030H, 000H, 030H, 000H	;
DB	06CH, 06CH, 06CH, 000H, 000H, 000H, 000H, 000H	;
DB	06CH, 06CH, 0FEH, 06CH, 0FEH, 06CH, 06CH, 000H	;
DB	030H, 07CH, 0C0H, 078H, 00CH, 0F8H, 030H, 000H	;
DB	000H, 0C6H, 0CCH, 018H, 030H, 066H, 0C6H, 000H	;
DB	038H, 06CH, 038H, 076H, 0DCH, 0CCH, 076H, 000H	;
DB	060H, 060H, 0C0H, 000H, 000H, 000H, 000H, 000H	;
DB	018H, 030H, 060H, 060H, 060H, 030H, 018H, 000H	;
DB	060H, 030H, 018H, 018H, 018H, 030H, 060H, 000H	;
DB	000H, 066H, 03CH, 0FFH, 03CH, 066H, 000H, 000H	;
DB	000H, 030H, 030H, 0FCH, 030H, 030H, 000H, 000H	;
DB	000H, 000H, 000H, 000H, 000H, 030H, 030H, 060H	;
DB	000H, 000H, 000H, 0FCH, 000H, 000H, 000H, 000H	;
DB	000H, 000H, 000H, 000H, 000H, 030H, 030H, 000H	;
DB	006H, 00CH, 018H, 030H, 060H, 0C0H, 080H, 000H	;
DB	07CH, 0C6H, 0CEH, 0DEH, 0F6H, 0E6H, 07CH, 000H	;0
DB	030H, 070H, 030H, 030H, 030H, 030H, 0FCH, 000H	;1
DB	078H, 0CCH, 00CH, 038H, 060H, 0CCH, 0FCH, 000H	;2
DB	078H, 0CCH, 00CH, 038H, 00CH, 0CCH, 078H, 000H	;3
DB	01CH, 03CH, 06CH, 0CCH, 0FEH, 00CH, 01EH, 000H	;4
DB	0FCH, 0C0H, 0F8H, 00CH, 00CH, 0CCH, 078H, 000H	;5

DB	038H,060H,0C0H,0F8H,0CCH,0CCH,078H,000H	;6
DB	0FCH,0CCH,00CH,018H,030H,030H,030H,000H	;7
DB	078H,0CCH,0CCH,078H,0CCH,0CCH,078H,000H	;8
DB	078H,0CCH,0CCH,07CH,00CH,018H,070H,000H	;9
DB	000H,030H,030H,000H,000H,030H,030H,000H	;
DB	000H,030H,030H,000H,000H,030H,030H,060H	;
DB	018H,030H,060H,0C0H,060H,030H,018H,000H	;
DB	000H,000H,0FCH,000H,000H,0FCH,000H,000H	;
DB	060H,030H,018H,00CH,018H,030H,060H,000H	;
DB	078H,0CCH,00CH,018H,030H,000H,030H,000H	;
DB	07CH,0C6H,0DEH,0DEH,0DEH,0C0H,078H,000H	;
DB	030H,078H,0CCH,0CCH,0FCH,0CCH,0CCH,000H	;
DB	0FCH,066H,066H,07CH,066H,066H,0FCH,000H	;
DB	03CH,066H,0C0H,0C0H,0C0H,066H,03CH,000H	;
DB	0F8H,06CH,066H,066H,066H,06CH,0F8H,000H	;
DB	0FEH,062H,068H,078H,068H,062H,0FEH,000H	;
DB	0FEH,062H,068H,078H,068H,060H,0F0H,000H	;
DB	03CH,066H,0C0H,0C0H,0CEH,066H,03EH,000H	;
DB	0CCH,0CCH,0CCH,0FCH,0CCH,0CCH,0CCH,000H	;
DB	078H,030H,030H,030H,030H,030H,078H,000H	;
DB	01EH,00CH,00CH,00CH,0CCH,0CCH,078H,000H	;
DB	0E6H,066H,06CH,078H,06CH,066H,0E6H,000H	;
DB	0F0H,060H,060H,060H,062H,066H,0FEH,000H	;
DB	0C6H,0EEH,0FEH,0FEH,0D6H,0C6H,0C6H,000H	;
DB	0C6H,0E6H,0F6H,0DEH,0CEH,0C6H,0C6H,000H	;

```

DB      038H,06CH,0C6H,0C6H,0C6H,06CH,038H,000H      ;
DB      0FCH,066H,066H,07CH,060H,060H,0F0H,000H      ;
DB      078H,0CCH,0CCH,0CCH,0DCH,078H,01CH,000H      ;
DB      0FCH,066H,066H,07CH,06CH,066H,0E6H,000H      ;
DB      078H,0CCH,0E0H,070H,01CH,0CCH,078H,000H      ;
DB      0FCH,0B4H,030H,030H,030H,030H,078H,000H      ;
DB      0CCH,0CCH,0CCH,0CCH,0CCH,0CCH,0FCH,000H      ;
DB      0CCH,0CCH,0CCH,0CCH,0CCH,078H,030H,000H      ;
DB      0C6H,0C6H,0C6H,0D6H,0FEH,0EEH,0C6H,000H      ;
DB      0C6H,0C6H,06CH,038H,038H,06CH,0C6H,000H      ;
DB      0CCH,0CCH,0CCH,078H,030H,030H,078H,000H      ;
DB      0FEH,0C6H,08CH,018H,032H,066H,0FEH,000H      ;
DB      078H,060H,060H,060H,060H,060H,078H,000H      ;
DB      0C0H,060H,030H,018H,00CH,006H,002H,000H      ;
DB      078H,018H,018H,018H,018H,018H,078H,000H      ;
DB      010H,038H,06CH,0C6H,000H,000H,000H,000H      ;
DB      000H,000H,000H,000H,000H,000H,000H,0FFH      ;

DB      030H,030H,018H,000H,000H,000H,000H,000H      ;
DB      000H,000H,078H,00CH,07CH,0CCH,076H,000H      ;
DB      0E0H,060H,060H,07CH,066H,066H,0DCH,000H      ;
DB      000H,000H,078H,0CCH,0C0H,0CCH,078H,000H      ;
DB      01CH,00CH,00CH,07CH,0CCH,0CCH,076H,000H      ;
DB      000H,000H,078H,0CCH,0FCH,0C0H,078H,000H      ;
DB      038H,06CH,060H,0F0H,060H,060H,0F0H,000H      ;

```

```

DB      000H,000H,076H,0CCH,0CCH,07CH,00CH,0F8H      ;
DB      0E0H,060H,06CH,076H,066H,066H,0E6H,000H      ;
DB      030H,000H,070H,030H,030H,030H,078H,000H      ;
DB      00CH,000H,00CH,00CH,00CH,0CCH,0CCH,078H      ;
DB      0E0H,060H,066H,06CH,078H,06CH,0E6H,000H      ;
DB      070H,030H,030H,030H,030H,030H,078H,000H      ;
DB      000H,000H,0CCH,0FEH,0FEH,0D6H,0C6H,000H      ;
DB      000H,000H,0F8H,0CCH,0CCH,0CCH,0CCH,000H      ;
DB      000H,000H,078H,0CCH,0CCH,0CCH,078H,000H      ;
DB      000H,000H,0DCH,066H,066H,07CH,060H,0F0H      ;
DB      000H,000H,076H,0CCH,0CCH,07CH,00CH,01EH      ;
DB      000H,000H,0DCH,076H,066H,060H,0F0H,000H      ;
DB      000H,000H,07CH,0C0H,078H,00CH,0F8H,000H      ;
DB      010H,030H,07CH,030H,030H,034H,018H,000H      ;
DB      000H,000H,0CCH,0CCH,0CCH,0CCH,076H,000H      ;
DB      000H,000H,0CCH,0CCH,0CCH,078H,030H,000H      ;
DB      000H,000H,0C6H,0D6H,0FEH,0FEH,06CH,000H      ;
DB      000H,000H,0C6H,06CH,038H,06CH,0C6H,000H      ;
DB      000H,000H,0CCH,0CCH,0CCH,07CH,00CH,0F8H      ;
DB      000H,000H,0FCH,098H,030H,064H,0FCH,000H      ;
DB      01CH,030H,030H,0E0H,030H,030H,01CH,000H      ;
DB      018H,018H,018H,000H,018H,018H,018H,000H      ;
DB      0E0H,030H,030H,01CH,030H,030H,0E0H,000H      ;
DB      076H,0DCH,000H,000H,000H,000H,000H,000H      ;
DB      000H,010H,038H,06CH,0C6H,0C6H,0FEH,000H      ;

```



```
%endif
```

```
DIAG_TEST:
```

```
    IN     AL,IOBYTE           ;Start at Reset location, few bytes available to FFFFFH
    CMP    AL,03FH             ;If 3F then diagnostic, '?' continously of CRT
    JZ     XLOOP
    JMP    INIT                ;(Start of this monitor)
```

```
XLOOP: NOP                    ;For hardware address lines debug viewing
```

```
    XOR    BX,BX
    MOV    DS,BX               ;Read even & odd byte from RAM
```

```
    MOV    AL,[BX]
    MOV    AL,[BX+1]
    MOV    AX,word [BX]
    MOV    AX,word [BX+1]
```

```
    IN     AL,IOBYTE           ;see if we need a different character to output (checks I/O)
    OUT    01H,AL
    CMP    AL,01FH             ;If 1F within this loop then go to RAM at 500H
    JNZ    XLOOP
    JMP    word 0H:500H         ;Jump to here in low RAM
```

```
DB      '<--- END OF 8086/8088 Monitor V10.33 (John Monahan, 7/26/2014)  '
```

```
%if     MONITOR_ROM
```

```
TIMES 0FFF0H-($-$$) DB 0
```

```
JMP     word 0F000H:DIAG_TEST      ;Not clear why but need long jump for board hardware
```

```
TIMES 0FFFEH-($-$$) DB 0
```

```
DB      0FCH                      ;"Model Number" IBM PC/AT (At FFFEh)
```

```
DB      0H                        ;Skip Checksum
```

```
%endif
```

```
;----- LOW RAM VARIABLES (Used mainly by PC-BIOS section) -----
```

```
absolute 2H*4
```

```
NMIInt:   resw      2              ;Non-maskable interrupt location (8H)
```

```
absolute 5H*4
```

```
PrintScreen: resw      2          ;Print Screen function
```

```
absolute 8H*4          ;Location for our hardware interrupts (20H, Same as IBM-PC hardware)
```

```
Start8259A_Ints  resw      2      ;      8      Timer Tic          TIMER          \
                                     ;      9      Keypressed        KEYHND          \
```

```

;   A   Reserved           DUMMY_RETURN   \
;   B   Comm Hardware      DUMMY_RETURN   \Normal location for
;   C   Comm Hardware      DUMMY_RETURN   /IBM hardware interrupts
;   D   Disk Hardware      DUMMY_RETURN   /
;   E   Diskette Hardware  DUMMY_RETURN   /
;   F   Printer Hardware   DUMMY_RETURN   /

```

```

absolute      10H*4
CRTINT        resw      2          ;Software interrupt used in this BIOS (and IBM-PC/AT)

```

```

absolute      13H*4
MAIN_DISK_VEC resw      2          ;Disk (Hard & Floppy) software interrupt
absolute      (13H*4)+2
MAIN_DISK_SEG resw      2          ;Disk software interrupt segment (Normally this CS)

```

```

absolute      1DH*4          ;Pointer to CGA Video Board paramaters table
VID_PARM_PTR  resw      2
absolute      (1DH*4)+2
VID_PARM_PTR_SEG resw      2      ;Video Board Table segment (0 here)

```

```

absolute      1EH*4          ;Pointer to Floppy Disk paramaters table
FDISK_PARMS   resw      2      ;On MSDOS Boot, this points to the boot Floppy Disk Variables Table
absolute      (1EH*4)+2

```

```

FDISK_PARMS_SEG      resw    2          ;Disk Variables Table segment (Normally this CS)

                                absolute    1FH*4

EXT_CHAR_PTR  resw    2          ;7CH, For Graphics mode extra characters pointers
                                absolute    (1FH*4)+2

EXT_CHAR_PTR_SEG  resw    2          ;For Graphics mode extra characters pointers segment (Normally this CS)

                                absolute    40H*4

OLD_DISK_VEC  resw    2          ;Pointer to the original PC Floppy Disk Int Vector (relocated because of HDISK)
                                absolute    (40H*4)+2

OLD_DISK_SEG  resw    2          ;New Floppy Disk software interrupt segment (Normally this CS)

                                absolute    41H*4

HDISK_PARMS      resw        2          ;Pointer to HARD DISK #1 paramater table
                                absolute    (41H*4)+2

HDISK_PARMS_SEG      resw    2          ;HARD DISK paramater table segment (Normally this CS)

                                absolute    43H*4

EXT_CHAR_PTR2  resw        2          ;10CH, Pointer to Graphics Character set
                                absolute    (43H*4)+2

EXT_CHAR_PTR2_SEG  resw    2          ;Pointer to Graphics Character Set Segment

                                absolute    46H*4

HDISK2_PARMS  resw    2          ;Pointer to HARD DISK #2 paramater table

```

```

        absolute      (46H*4)+2

HDISK2_PARMS_SEG resw  2          ;HARD DISK paramater table segment (Normally this CS)


        absolute      400H        ;Low RAM data area (set the same as for IBM-PC BIOS)


RS232_BASE   resw  4          ;Addresses for RS232 Adaptors (if any)

PRINTER_BASE resw  4          ;Address of Printers (if any)


EQFLAG       resw  1          ;410H, equipment flag (two bytes)

MFG_TST      resb  1          ;412H, MFG initialization flag (not used)

memrsz       resw  1          ;413H, memory size (kilobytes)

expmam       resw  1          ;415H, expansion ram size


KB_FLAG       resb  1          ;417H (Insert, shift etc. flags)

KB_FLAG_1    resb  1          ;418H, Second byte of keyboard status

chrcnt       resb  1          ;419H, characters in buffer (Alt_Keypad on PC)


bufhd        resw  1          ;keyboard buffer head      (40:1AH)

buftl        resw  1          ;keyboard buffer tail      (40:1CH)

keybuff      resw  16         ;keyboard data buffer      (40:1EH)

kbend        resw  1          ;end of buffer

chrmax equ    32             ;buffer length

; \

```

```

; \
; Keyboard buffer area
; /
;/

```

```
absolute    43EH
```

```

SEEK_STATUS  resb    1          ;Seek status  (40:3EH)
CURRENT_HEAD resb    1          ;On IBM PC, motor status (40:3FH)
CURRENT_DRIVE resb    1          ;On IBM PC, motor count (40:40H)

IBM_DISK_STATUS  resb    1          ;Returned disk status  (40:41H)

;

DMA_OFFSET  resw    1          ;DMA offset address for controller  (On PC this area is used by FDC)
DMA_SEGMENT resw    1          ;DMA segment address for controller
CURRENT_SECTOR  resb    1
CURRENT_TRACK  resb    1
CURRENT_TRACK_HIGH resb    1

```

```
absolute    449H          ;Video board paramaters
```

```

CRT_MODE      resb    1          ;449H, Video Display Mode
; 0 = 40 x 25 text (no color)
; 1 = 40 x 25 text (16 color)

```

```

; 2 = 80 x 25 text (no color)
; 3 = 80 x 25 text (16 color)
; 4 = 320 x 200 graphics 4 color
; 5 = 320 x 200 graphics 0 color
; 6 = 640 x 200 graphics 0 color
; 7 = 80 x 25 text (mono card)

```

```

CRT_COLS      resw  1
CRT_LEN              resw  1
CRT_START      resw  1
CURSOR_POSN     resw  8      ;IBM has 8

CURSOR_MODE     resw  1      ;446H, Cursor shape
ACTIVE_PAGE     resb  1
ADDR_6845       resw  1
CRT_MODE_SET     resb  1
CRT_PALETTE     resb  1

IO_ROM_INIT     resw  1      ;467H, Anchor location to implement extra ROMS
IO_ROM_SEG      resw  1      ;469H
INTR_FLAG       resb  1

timlow          resw  1      ;timer low count (40:6CH) for timer
timhi           resw  1      ;timer high count
timofl          resb  1      ;timer overflow flag

```


absolute	48BH		;Disk (Floppy & HDisk) data control area
			;48B-495H
LAST_RATE	resb	1	;488H, Addditional Floppy data
HF_STATUS	resb	1	;48CH, Additional HDisk data area
HF_ERROR	resb	1	
HF_INT_FLAG	resb	1	
HF_CNTRL	resb	1	
DSK_STATE	resw	2	;490H, Additional Diskette Area (must be 0 for MS-DOS 4.01)
DSK_TRK	resb	3	
KB_FLAG_2	resb	1	;497H, Last keyboard LED/Shift key state (on AT)
			;498H, RTC additional data area
USER_FLAG	resw	1	; offset address of user wait flag
USER_FLAG_SEG	resw	1	; segment " " "
RTC_LOW	resw	1	; user RTC low word
RTC_HIGH	resw	1	; user RTC high word
RTC_WAIT_FLAG	resb	1	;4A0H, user wait active
absolute	4A1H		;4A1H-4A7H Reserved for LAN bytes
absolute	4A8H		;4A8H-4ABH, Segment:Offset address of video parameter control block
absolute	4ACH		;Reserved

```

        absolute    4F0H        ;Inter-applications communications area. I will use it for this BIOS

CONSOLE_FLAG  resw  1          ;0000 if MSDOS output to Propeller board, 0001 to LAVA board, 0002 to VGA Video board.
ZFDC_ERR_CODE resb  1          ;Error code returned by ZFDC controller in AH for error
DEBUG_FLAG    resb  1          ;If not Zero display track, side, sector etc info during disk R/W
ZFDC_INIT_FLAG resb  1          ;Flag to indicate ZFDC board has been initialised
SECTORS_TO_DO resb  1          ;Number of sectors to transfer in current operation
SECTORS_DONE  resb  1          ;Number actually transferred
VERIFY_FLAG   resb  1          ;0 for normal sector reads, NZ, if just sector verifys required
ES_STORE      resw  1          ;Used for INT 10H, String write

        absolute    500H        ;To keep things the same as for the IBM_PC
STATUS_BYTE   resb  1          ;Flag for print screen routine
SINGLE_DRIVE   resb  1          ;

        absolute    510H        ;This area is used by BASIC. However MSDOS will not boot if used

        absolute    522H        ;To keep things the same as for the IBM_PC
DOS_INIT_AREA resb  16          ;IBMBIO.COM buffers the directory of the boot
                                   ;device here at load time when locating the guts
                                   ;of the operating system (IBMDOS.COM/MSDOS.SYS)

        absolute    530H        ;To keep things the same as for the IBM_PC
DOS_MODE_AREA resb  4

        absolute    534H        ;To keep things the same as for the IBM_PC

```

```
MORE_DOS      resb    12
```

```
absolute      0B800H      ;IBM-PC Color Video Board RAM area
```

```
Video_Ram      resw    16384 *2      ;Video RAM area (IF, Lomas or other S100 PC Video board is used)
```

```
absolute    7c00h        ;0000:7c00H
```

```
DOS_BOOT_LOC: resw    1    ;----MS-DOS/FreeDOS BOOT LOCATION
```

```
absolute      7c00h+510          ;0000:7dfeH
```

```
DOS_BOOT_SIGNATURE: resw    1    ;<---MS-DOS Valid Boot Signature Location (0AA55H)
```

```

;Remember there is a high RAM data area used (only) for the IDE Board
;diagnostic functions. These variables will normally be accessed as SS:[BP]
;This is used by the IDE drive diagnostic commands ONLY. We need an area
;to store buffers in RAM. In a full 1MG system they will at D000:E000H
;Remember also the stack is normally at D000:FFFCH

```

```
absolute    0E000H    ;For SS:BP -> D000:E000H
```

```
RAM_DRIVE_SECresw      1          ;This area will be in top of RAM well below stack (used by IDE Board sections)
```

```

RAM_DRIVE_TRK resw      1
RAM_DRIVE_HEAD      resw      1
RAM_DRIVE_COUNT      resw      1
RAM_SEC              resw      1
RAM_TRK              resw      1
DELAYStore          resw      1
RAM_DMA              resw      1
RAM_DMA_STORE resw      1
SECTOR_COUNT resw      1
CURRENT_IDE_DRIVE resw      1
DISPLAY_FLAG resw      1

```

```

absolute      0E100H      ;For SS:BP -> D000:E000H

```

```

IDE_Buffer      resb      200H      ;512 Byte buffer for IDE Sector R/W
IDE_Buffer2      resb      200H      ;512 Byte buffer for IDE Sector Verify

```

```

;End of 8086 Monitor

```